2020-09

GUIDANCE, NAVIGATION, AND CONTROL OF A QUADROTOR DRONE WITH PID CONTROLS

DeBock, Sandra

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

GUIDANCE, NAVIGATION, AND CONTROL OF A QUADROTOR DRONE WITH PID CONTROLS

by

Sandra DeBock

September 2020

Thesis Advisor:	Xiaoping Yun
Second Reader:	James Calusdian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
GUIDANCE, NAVIGATION, AND CONTROL OF A QUADROTOR DRONE WITH PID CONTROLS

5. FUNDING NUMBERS

6. AUTHOR(S) Sandra DeBock

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

The threat of drones in sensitive airspace is a growing issue. Drone detection currently focuses on security threats of undesired surveillance. In this thesis, we seek to determine whether a supervisory controller applied to a quadrotor drone will suffice as a feasible option for an autonomous drone. This is achieved by applying a proportional and derivative control law, programmed within MATLAB, to a pre-built simulation model and implemented on the Parrot Mambo Drone for experimental flights. The analysis is accomplished by comparing the performance and accuracy of the simulated trial flight, an experimental flight on the Parrot Mambo Drone with no altitude change, and an experimental flight with adjustments in all six degrees of freedom on the Parrot Mambo Drone. The results show that with a supervisory controller applied to a quadrotor drone, the drone can perform the desired tasks autonomously at a higher standard than without a supervisory controller applied. The proportional and derivative controls implemented on the Parrot Mabo Drone for the experimental flights have the best performance of the control law investigated.

14. SUBJECT TERMS guidance, navigation, control, quadrotor drone, intercept	**15. NUMBER OF PAGES** 139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

GUIDANCE, NAVIGATION, AND CONTROL OF A QUADROTOR DRONE WITH PID CONTROLS

Sandra DeBock
Captain, United States Marine Corps
BS, San Diego State University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: Xiaoping Yun
Advisor

James Calusdian
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The threat of drones in sensitive airspace is a growing issue. Drone detection currently focuses on security threats of undesired surveillance. In this thesis, we seek to determine whether a supervisory controller applied to a quadrotor drone will suffice as a feasible option for an autonomous drone. This is achieved by applying a proportional and derivative control law, programmed within MATLAB, to a pre-built simulation model and implemented on the Parrot Mambo Drone for experimental flights. The analysis is accomplished by comparing the performance and accuracy of the simulated trial flight, an experimental flight on the Parrot Mambo Drone with no altitude change, and an experimental flight with adjustments in all six degrees of freedom on the Parrot Mambo Drone. The results show that with a supervisory controller applied to a quadrotor drone, the drone can perform the desired tasks autonomously at a higher standard than without a supervisory controller applied. The proportional and derivative controls implemented on the Parrot Mabo Drone for the experimental flights have the best performance of the control law investigated.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

6-DoF	six-degrees-of freedom
IMU	inertial measurement unit
LQR	linear quadratic regulator
MPC	model predictive control
NED	North-East-Down frame
PID	proportional-integral-derivative
PD	proportional and derivative
PMD	Parrot Mambo Drone
RNDIS	Remote Network Driver Interface Specification
SDK	software development kit
SSPPM	Simulink Support Package for Parrot Minidrones
UAV	unmanned aerial vehicle
WP	waypoint

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I want to thank my thesis advisors, Dr. Xiaoping Yun and Dr. James Calusdian. Dr. Yun, thank you for your invaluable advice and patience with me throughout this entire thesis writing process. Dr. Calusdian, thank you for your guidance in framing my focus and helping me through with discussions when I lost that focus.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In both military and civilian life, unmanned aerial vehicles (UAV) have become increasingly common and necessary. Current forms of micro UAVs rely heavily on human interaction for flight. For example, to obtain an accurate and immediate response to a micro UAV regarding its destination a ground-based controller is commonly used. However, this reliance on a ground-based controller creates a significant constraint on the tasks and capabilities the micro UAV can perform. One such constraint is the restricted distance that the micro UAV may travel. The operational range of a micro UAV may be restricted by the line of sight distance from the ground controller. Additionally, the reliance on human operation may not allow the optimized performance and flight efficiency that autonomous operation could provide. Autonomous UAVs have become more of an emphasis in research and industry. There are many industries that use autonomous drones to optimize industry output. These drones are expensive and, in most cases, large. The further development of this innovation in smaller UAVs will allow for a more intuitive and agile UAV that can perform tasks without the need of a ground-based controller. In this chapter the background, motivation, literature review, and objectives for this thesis research are presented.

A. BACKGROUND AND MOTIVATION

The use of quadrotor drones has become a common part of modern society. The United States Federal Aviation Administration currently defines any size UAV as simply "a UAV." From this definition, a quadrotor drone is defined as a UAV and is generally controlled either by a remote control guided by a human or autonomously by an onboard computer. The distinction between the two types of control are that an autonomous drone does not require any human interaction or intervention to carry out an assigned mission.

With the advances in the technology of sensors, control theory, and aerodynamic knowledge, UAVs have become affordable and practical for many uses. The most common UAVs on the market are generally under one meter in size and have the advantage of being portable. The size is a selling point, but also creates issues with

stability when there are unpredictable environmental conditions. Another issue with smaller drones is that the sensors designed for the platform are smaller in size causing potential undesired noise, vibrations, and temperature increases within the compact housing. These focus areas have been well documented and are continuously being perfected and improved upon, as in [1], [2], [3], [4], and [5].

Autonomous micro UAVs have the portability and potential to open opportunities for the operator to use a drone to accomplish a mission without having to have constant visual and control of where and what the drone does. Autonomous drones that can seek out a specific object would greatly benefit many sectors in both the civilian area and military divisions.

B. LITERATURE REVIEW AND RELATED WORKS

Quadrotor drones are one of the most popular aerial vehicles due to their simplicity and quick response time. A wide number of studies have resulted in a flight command that allows the platform to follow a trajectory. The most common form of control is through a proportional-integral-derivative (PID) controller. This form of control uses the referenced input of where the drone is meant to be and compares it to the actual measured state. From this comparison it determines the error and seeks to minimize it to reach its goal. There has been further research that has implemented a Fuzzy PID, detailed in [6].

Another form of control involves a linear quadratic regulator (LQR) developed in [7]. With this method, the system obtains a feedback signal for the linear system. This method is determined by minimizing the cost function of a quadratic form. The appropriate minimum performance index is the optimal system parameter value. The algorithm of the LQR control system is designed so that all the states from the past can both be tracked and developed, as in [8], [9], and [10].

The use of UAVs has become a norm for modern society. Allen in [5] has done extensive research on the civilian side of what companies are doing now. The author delves into the company Guard From Above that uses birds to capture rouge drones. Altitude and position control of a Parrot Mambo Mini drone with the use of a PID in [6]

has been developed to determine if the control measures of a PID control law will benefit the drone and the performance output. Another form of research that has been analyzed in the development of UAV control methods is from [1] using a UAV to follow railroad tracks to pinpoint any issues with a section of the railroad path. Other researches have compared the performance standards of different types of control implemented on the same UAV for comparison, as explained in [3].

C. UAV SELECTION

For this research, it is important to ensure that the quadrotor drone chosen for trials will fit the following criteria: programable with a software package (MATLAB, ROS etc.), cost effective for practical future use, durable in order to deal with potential collisions, small in size to allow for easy transport, and adaptable from a laboratory setting to a natural real world environment.

There are many companies that design and sell small UAVs. After analysis of drone style, it is determined that a quadrotor drone will give the best overall quality for stability and robustness. This is discussed in further detail within Chapter III of this research. The top quadrotor drone companies on the market that currently meet the basic requirements are: DJI, Kespry, and Parrot. Of the three, Parrot is chosen for its compatibility with MATLAB and overall low cost. The price for a Parrot Mambo Drone (PMD) is a third of the other brands, as explained in [11]. The models from other companies did project a longer flight time and a more robust video platform, but these factors are not needed for this research.

D. THESIS OUTLINE AND OBJECTIVES

In this research, the goal is to determine if a supervisory controller can be used to provide autonomous operation of a quadrotor drone. The use of control measures based on PID control laws will be investigated, along with the gain values for the controller that will be determined through a process of trial and error to achieve the desired parameters for the research.

The applicable background in control laws, motivation, and previous work are given in this chapter. The chosen quadrotor drone and the reasons behind the choice are also discussed within Chapter I.

Next, the dynamic model for a simulated and experimental flight of a quadrotor drone is presented in Chapter II. The coordinate systems for each model parameter are defined, and the equations of motion that are necessary to analyze the flight path of the quadrotor drone are detailed.

The software and hardware that are required to analyze the data and perform the flight tests for this research are presented in Chapter III. Each system is explained and defined for the particular use it has for the research. The implementation of how all systems work together as one to produce the required data for the research is also explained.

The framework for the control design is described in Chapter IV. The control law that is implemented, along with other options for control, is derived in detail and explained. The implementation of the control law used for the simulated and experimental flight is presented.

Chapter V of the research depicts the Simulink model that is used and detailed in the subsections that are adapted from the original model. After the model is explained, the results for the simulated flight are presented.

The MATLAB code and control logic used are explained in Chapter VI. The MATLAB code contains the control laws that are presented in the earlier chapter. The experimental results are obtainable at the end of the chapter.

Finally, in Chapter VII, a conclusion based on the simulated and experimental results is provided. Also included are both a summary of the limitations encountered during the research and subject areas for future work.

II. DYNAMIC MODELS

The ability to control a UAV is a challenging task for the following reasons. The Parrot Mambo Drone (PMD) is a non-linear, multi-input, multi-output, and under-actuated system. The system is only controlled by four actuators and is tracked within another system that does not share the same reference frame as the PMD. The purpose of this chapter is to provide a foundation for the equations of motion used to simulate the quadrotor flight paths. This chapter begins by defining the assumptions made to determine the dynamic model using Newton-Euler formalism presented in [12]. With these assumptions, the definition and explanation of the required reference frames to simulate flight are developed. Once the reference frames are defined, the equations of motion for the PMD dynamic model are detailed.

A. MODELING ASSUMPTIONS

To accurately use Newton-Euler formulation [12], and to analyze the PMD in motion, certain assumptions must be made:

- The drone acts as a symmetric rigid body with the center of mass and body frame both fixed at the same location.
- The propellers are a rigid structure throughout flight.
- The battery provides a constant voltage and current to the drone throughout each flight.
- For each flight, the waypoints and position are always known.

B. COORDINATE SYSTEM AND REFERENCE FRAMES

The coordinate system is a geometric analysis that describes what an object is doing within its specific reference frame. It uses coordinates to determine the position of an element in a space, as explained in [13]. The order of the coordinates is important, and they are identified by their positions on the axis.

The three primary axes and three rotation angles are depicted in Figure 1. The following section will identify the appropriate reference and axis alignment to ensure proper flight of the PMD.

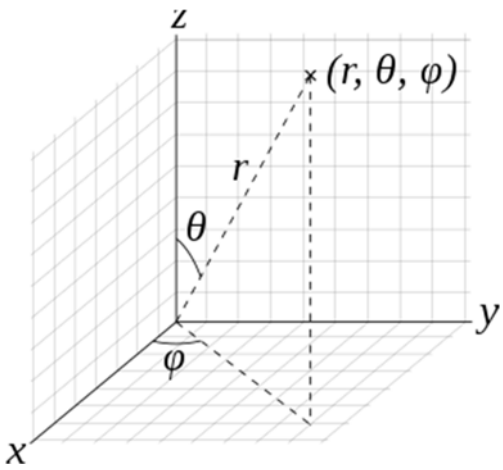

Figure 1. Coordinate system orientation and rotation angle depiction. Source: [13].

1. Drone Reference Frames

The two major reference frames used to obtain the equations of motion for the PMD are the body reference frame and the inertial reference frame. The PMD body reference is located at the PMD center of mass. All three axes start at the center of mass and point to their respective directions. The axes are defined by X, Y, and Z, and are labeled x_b, y_b, and z_b to annotate body frame. The X-axis is in the direction along the nose. The Y-axis points to the right side of the PMD and the Z-axis points downward following the right-hand rule with a positive rotation to the right presented in [11].

The second reference frame required is the inertial reference frame. This is a reference frame fixed on the surface of the Earth. For this reference frame, the positive X-axis points true North, the Y-axis points true East with the Z-axis pointing down, and perpendicular to the surface of the Earth. This is referred to as the North-East-Down frame (NED), depicted in Figure 2.

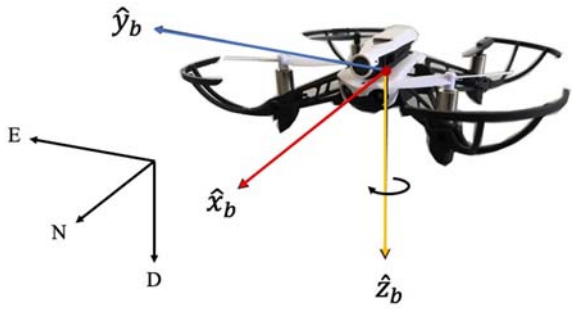

Figure 2. The PMD body fixed reference frame

2. OptiTrack Coordinate System

The OptiTrack system is designed with a global coordinate system depicted in Figure 3. This model is designed with the Y-axis oriented in the vertical direction, the X-axis to the left, and Z-axis in the forward direction depicted in [4].

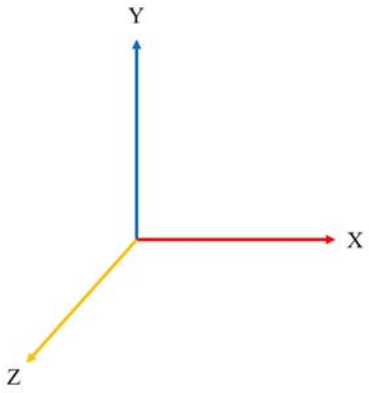

Figure 3. OptiTrack original coordinate system

For this research, the OptiTrack is used as the inertial frame while flights are conducted. The system presents a unique opportunity to adapt the OptiTrack reference frame to meet that of the PMD. To orient the data sent from OptiTrack to MATLAB, a series of functions and codes are used to adjust the rotation of the OptiTrack reference frame. The new orientation matches with the NED of the PMD. Utilizing the MATLAB function *optitrack_getDronePose*, provided in Appendix C, the OptiTrack data is converted into the preferred orientation of the PMD using the following procedures:

- The OptiTrack quaternion is rotated by: $\left[\cos(\pi/4) \quad \sin(\pi/4) \quad 0 \quad 0\right]$.
- The Euler angles for the rigid body, to establish the quaternion with u_1 through u_4, are normalized as the vector u .
- The new rotation orientation is established with MATLAB function *rotate_v_by_q*, listed in Appendix F, using the normalized u and new quaternion rotation.
- The angle orientation of $\left[\phi \quad \theta \quad \psi\right]$ with MATLAB command *quat2eul* [14] is established.
- New $\left[X \quad Y \quad Z\right]$ axis positions are defined.
- New $\left[\phi \quad \theta \quad \psi\right]$ angles are defined.
- The output for the final position array for tracking the PMD is given by: $\left[X \quad Y \quad Z \quad \phi \quad \theta \quad \psi\right]$.

C. EQUATIONS OF MOTION

The PMD body reference frame can assume any position with any orientation with respect to the inertial frame. The PMD orientation and position are based on the inertial reference frame. To determine the orientation, the Euler rotations are used to relate the two frames. The order that the rotations occur is essential to determine the PMD body reference frame correctly. A representation of the inertial frame i and the body frame b to help orient the movement of the object is represented in Figure 4.

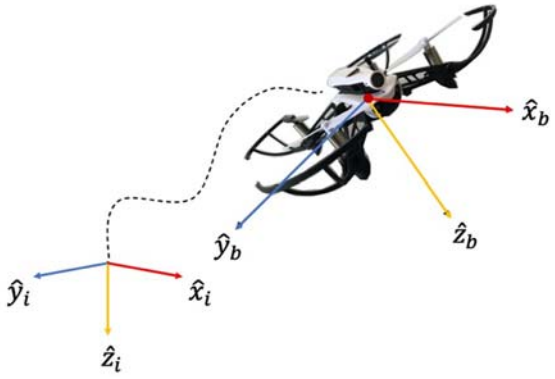

Figure 4. Body reference frame with respect to the Earth inertial reference frame

The PMD dynamics are conveyed in the body frame with external forces applied to the center of mass and can be formulated from [12] with Newton-Euler as

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} mI & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{\Omega} \end{bmatrix} + \begin{bmatrix} \Omega \times m\dot{p} \\ \Omega \times I\dot{\Phi} \end{bmatrix}. \tag{1}$$

Behavior of the PMD is described with four outputs and two inputs. The inputs are the applied forces F_i defined in [12] by

$$F_i = b(\omega_i)^2, \tag{2}$$

and torques τ_i

$$\tau_i = d * F_i. \tag{3}$$

In the above, i refers to the rotor position from one through four of each motor, b is the static thrust constant, d is the constant that relates the rotor thrust and moment, and ω is the anugular velocity of the motor. These inputs are generated by the four motors on the PMD which determine the control inputs U_i. The length L is the distance between the center of mass and each rotor blade, and the mass m of the PMD is a

constant value. Since the PMD has a constant mass, symmetric configuration, and a fixed axis, the inertia matrix I is given by [10]

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{4}$$

To determine the control inputs of yaw, pitch, roll, and the total thrust, a combination of forces is used in conjunction with specific motors and rotors explained in [12]

$$U_1 = F_1 + F_2 + F_3 + F_4. \tag{5}$$

Depicted in Figure 5 is the directional rotations of each rotor on the PMD. The total thrust from the PMD is calculated by summing all of the forces on the PMD with (5).

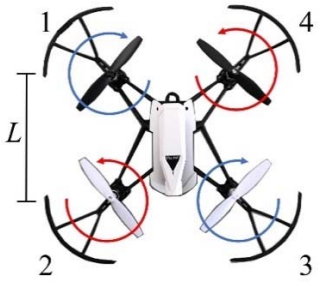

Figure 5. The PMD rotor rotations and motor placement

To establish roll ϕ , two of the rotors, 4 and 3, must slow while the opposite side rotors, 1 and 2, speed up to create the desired movement, as done in [12] using

$$U_2 = L(F_4 + F_2). \tag{6}$$

Pitch θ is presented in [12] and is achieved through increasing the speeds of rotors 1 and 4 and decreasing rotors 2 and 3

$$U_3 = L(F_1 + F_3). \tag{7}$$

Yaw ψ is calculated from the torque of the front and back rotors spinning counterclockwise, subtracted from the front rotors spinning clockwise, as utilized in [12]

$$U_4 = \tau_1 + \tau_2 - \tau_3 - \tau_4. \tag{8}$$

The four inputs form the vector U defined by [12]

$$U = [U_1 \quad U_2 \quad U_3 \quad U_4]^T. \tag{9}$$

The three rotations are restricted to ensure stable flight [6]:

- Roll angle $\left(-\pi/2 < \phi < \pi/2\right)$,
- Pitch angle $\left(-\pi/2 < \theta < \pi/2\right)$,
- Yaw angle $\left(-\pi < \psi < \pi\right)$.

The three rotations go through a sequence of the three angles. The first angle is ψ, which propagates from the original x-axis to form a new heading denoted with $\hat{x}_1$ to indicate it is the first of the Euler angles to be determined. Second is the rotation through θ, which moves from $\hat{x}_1$ to give a new heading towards $\hat{x}_2$. The final angle, ϕ, is created by the PMD rotating about the $\hat{x}_2$ axis, and settling to give the body reference frame comprised of $\hat{x}_b$, $\hat{y}_b$, and $\hat{z}_b$ [4]. The three rotations in a stepped sequence, to demonstrate what motion the PMD will assume in each phase, are referenced in Figure 6.

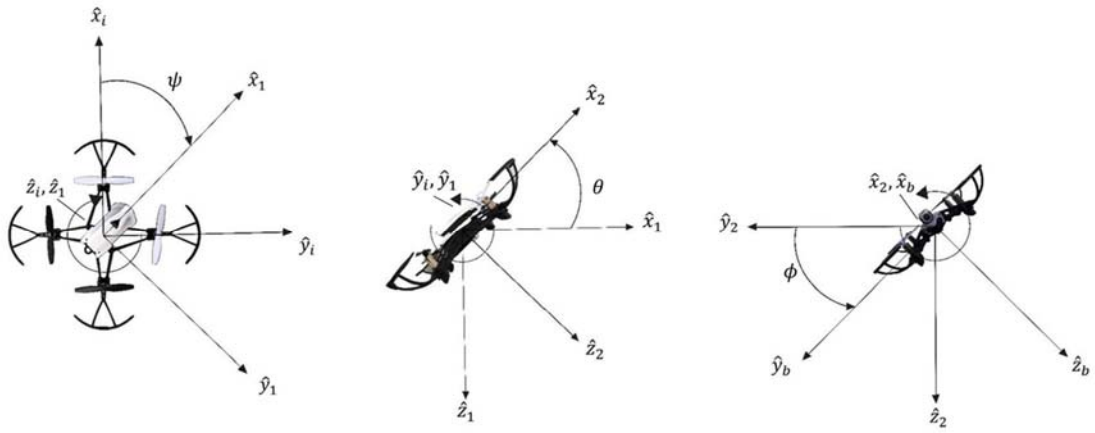

Figure 6. Euler rotations for the PMD orientation

A total of 12 states is used to characterize the PMD dynamic behavior: position $P = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}$, velocity $V = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$, Euler rotational angles $\Phi = \begin{bmatrix} \phi_x & \theta_y & \psi_z \end{bmatrix}$, and angular rate $\Omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}$, as seen in [12]. Using the Newton-Euler equations the state vector can be defined as a 12×1 vector

$$x(t) = \begin{bmatrix} P \\ V \\ \Phi \\ \Omega \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \\ \phi_x \\ \phi_y \\ \phi_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}. \tag{10}$$

The three-angles of rotation from the inertial frame to the body frame, depicted in Figure 6, can now be calculated with the use of direction cosine matrices adapted from [12] with $S = \sin$ and $C = \cos$ such that

$$R_{\psi} = \begin{bmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{z}_1 \end{bmatrix} = \begin{bmatrix} c\psi & s\psi & 0 \\ -s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{z}_i \end{bmatrix}, \tag{11}$$

$$R_{\theta} = \begin{bmatrix} \hat{x}_2 \\ \hat{y}_2 \\ \hat{z}_2 \end{bmatrix} = \begin{bmatrix} c\theta & 0 & -s\theta \\ 0 & 1 & 0 \\ s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{z}_1 \end{bmatrix}, \tag{12}$$

and

$$R_{\phi} = \begin{bmatrix} \hat{x}_b \\ \hat{y}_b \\ \hat{z}_b \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi & s\phi \\ 0 & -s\phi & c\phi \end{bmatrix} \begin{bmatrix} \hat{x}_2 \\ \hat{y}_2 \\ \hat{z}_2 \end{bmatrix}. \tag{13}$$

Combining the sequence of the three direction cosine matrices (11), (12), and (13) produces the complete conversion rotation matrix from the inertial and body frames, as seen in [10]

$$r_y = R_{\psi} R_{\theta} R_{\phi} \begin{bmatrix} \hat{x}_i \\ \hat{y}_i \\ \hat{z}_i \end{bmatrix}, \tag{14}$$

and

$$R = \begin{bmatrix} c\theta c\psi & -c\theta s\psi + s\phi s\theta c\psi & s\phi s\psi + c\phi s\theta c\psi \\ c\theta s\psi & c\phi c\psi + s\phi s\theta s\psi & -s\phi c\psi + c\phi s\theta s\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}. \tag{15}$$

To determine the time derivatives of (10) it is required to determine the Q , J , and *skew* matrices:

$$Q = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & s\phi c\theta \\ 0 & -s\phi & c\phi c\theta \end{bmatrix}, \tag{16}$$

$$J = \begin{bmatrix} J_x & 0 & 0 \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix}, \tag{17}$$

and

$$skew(\omega) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{18}$$

With the required information from (16), (17), and (18), the time derivatives of the state vector can be calculated with a few constant values: force due to the effect of gravity g, mass m of the PMD, torque τ that occurred from the four motors, and force due to thrust f_t. With these values adapted from [12], the time derivatives of (10) are defined by

$$\dot{p} = v, \tag{19}$$

$$\dot{v} = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} - R_{ib} \begin{bmatrix} 0 \\ 0 \\ f_t/m \end{bmatrix}, \tag{20}$$

$$\dot{\phi} = Q^{-1}\omega, \tag{21}$$

and

$$\dot{\omega} = J^{-1}\tau\left(-skew(\omega)J\omega\right). \tag{22}$$

The mathematical modeling of the dynamic behavior of the PMD is essential for this research to ensure a solid understanding of the PMD motion while operating.

III. SOFTWARE AND HARDWARE

To effectively analyze the flight data for this research, multiple software and hardware platforms are needed. The combinations of all components play a pivotal role in the analysis of the data. The software will be explained first to establish an understanding of the requirements needed for the hardware they support. Then the hardware and how each component is designed will be detailed. Finally, a brief description is offered in this chapter of what and how each system operates and contributes to the research.

A. SOFTWARE

1. MATLAB 2019b

This research relies on MATLAB 2019b for the programming platform to conduct the experimental runs. MATLAB is designed particularly for academic data analysis, algorithm development, and to create models and applications for scientist and engineers [15]. MATLAB is used to develop the control algorithm to manage the PMD flight dynamics along with collecting the data produced from the flights described within this research. With the collected data, MATLAB is used to create models to display the PMD flight details. Data is also imported from the supporting software to give additional analysis of the trajectory using the graphics functions in MATLAB.

2. Simulink

"Simulink is a platform for model-based design that supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems." [16] A developed Simulink model is used to simulate a desired flight path for this research. This software allows for simulated runs of a variation of values to pinpoint the desired response of the PMD. This is accomplished by manipulating the multiple subsystems within the Simulink model. The model is explained in full detail in Chapter V.

3. Motive: Tracker

Motive is a software that provides high-performance optical tracking through motion capture software using high-speed tracking cameras [16]. The software utilized within this research is Motive Tracker, which allows for real-time workflow by tracking the desired objects in six-degrees-of-freedom (6-DoF) [17]. Motive Tracker is capable of recording data in 2D, 3D, and Solved. Solved is a data product of the Motive Tracker software, and it provides positional and rotational, 6-DoF, tracking data of rigid bodies [18]. For the purposes of this research Solved data is used. To use Solved in the software, the geometry of the rigid body for the drone is identified with six reflective markers. The reflected markers are placed on the drone to ensure all sides of the PMD are seen by the OptiTrack cameras.

Depicted in Figure 7 and Figure 8 is the Solved model within the Motive program. For the experimental flights, the X, Y, and Z-axis positions, along with the Euler angles, are tracked on the quadrotor. Motive Tracker streams the data and uploads the data in MATLAB, through an ethernet connection, to process the desired data.

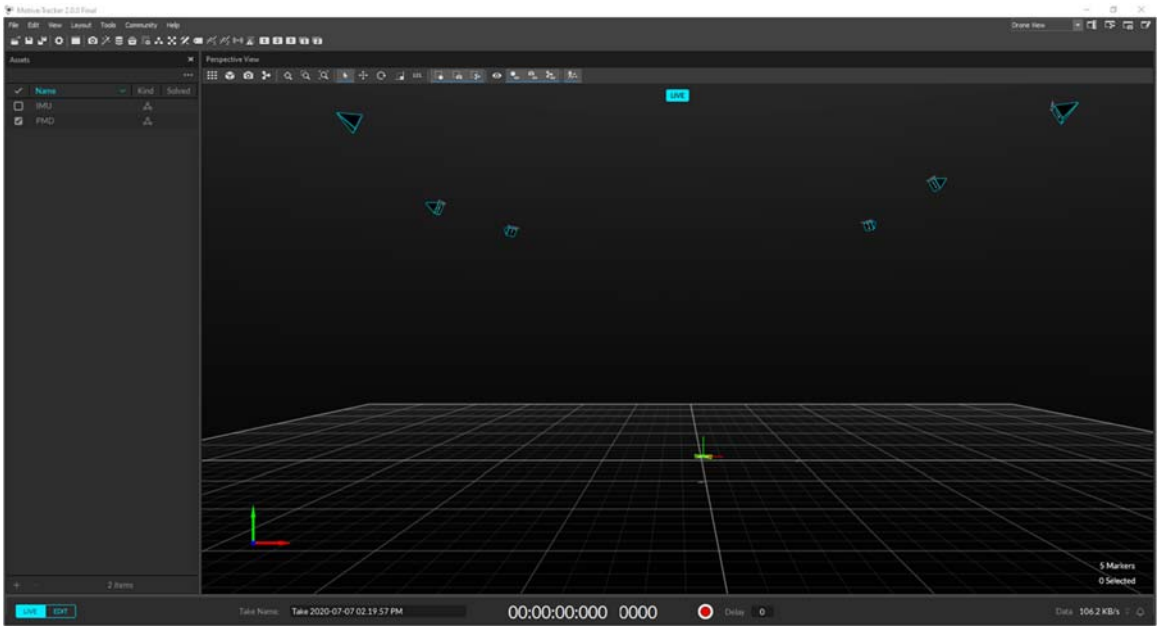

Figure 7. Motive Tracker of system set up for flights with all six cameras positioned

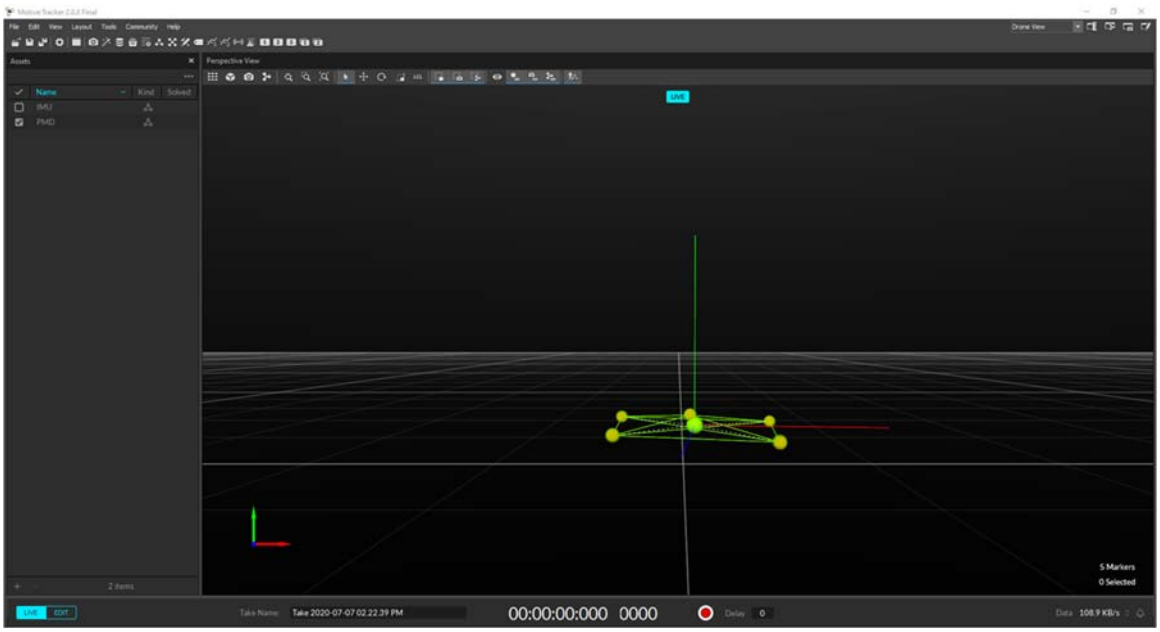

Figure 8. Motive Tracker of Solved data style for the flights with all six cameras positioned

B. HARDWARE

1. Parrot Mambo Drone

The PMD is chosen for its flexibility, affordability, and compatibility to MATLAB and Simulink. The quadrotor is equipped with an internal control system designed to stabilize itself for flight, making it a reliable platform to perform the experiments.

a. Drone Specifications

The PMD is equipped with a rigid body frame containing a battery, four sensors, four motors, four blades, and four bumpers depicted in Figure 9. The platform is both Wi-Fi and Bluetooth capable, allowing the PMD to be accessible to multiple forms of communication.

Figure 9. Parrot Mambo Drone

The rigid body measures 7.1×7.1 inches with bumpers on and weighs 73 grams. There is a USB connection to connect remote accessories on the PMD [11].

The battery used to power the PMD, pictured in Figure 10, is a 660 mAh Lithium polymer battery. The average run time on a full charge is 10 minutes. The charge time is 30 minutes with a 2 V 1A charger [11].

Figure 10. Parrot Mambo Drone battery pack

The four sensors are designed to ensure stability and safety of the PMD while being operated. On the base of the PMD is a grid-ultra sound sensor that measures the vertical distance. This measurement is accomplished by sending a high frequency sound pulse downward and then measuring how long it takes for the pulse to bounce off the floor and back to the sensor. For the PMD, this measurement is accurate to about 13 feet. The second sensor is a camera that takes 60 frames per second creating an optical flow. This sensor allows the PMD to determine how things move from one frame to the next by measuring the horizontal motion and speed. The third sensor is the pressure sensor inside the rigid body of the PMD structure. This sensor detects the changes in the air pressure as the PMD rises. The fourth sensor is the inertial measurement unit (IMU). This evaluates speed, tilt, and obstacle contact. This is accomplished with the use of three-axis

accelerometers to measure linear accelerations and three-axis gyros to measure the angular rates.

The four motors are each equipped with their respective rotors. The PMD comes with two colors of blades for easy identification when in flight. The design has the motors in an X direction configuration with opposing motors spinning as pairs depicted in Figure 11 [19]. These pairs spin in the same direction as each other, but opposite of the opposing pair. By coupling thrust and rotations, the system achieves roll, pitch, and yaw.

Figure 11. Configuration of the PMD in an X design

The PMD has an internal control system that has been designed to achieve the desired response from the drone for all 6-DoF to give the PMD the needed control to fly stably. This control is accomplished by having six PID controllers described in Chapter IV.

b. MATLAB Support Package for Parrot Drones

To pilot the PMD, the use of the MATLAB Support Package for Parrot Drones is utilized. This software development kit (SDK) provides an interface to control the PMD within MATLAB [20]. Multiple commands assist in the control of the PMD as well as commands to read data. The pre-programmed functions for the PMD are listed in Table 1 [21].

Table 1. Parrot Mambo Drone pre-programmed flight controls

Drone Navigation:	
takeoff	movedown
land	moveforward
abort	moveleft
flip	moveright
move	moveup
moveback	turn

Drone Flight Data:
readHeight
readOrientation (read Euler angles)
readSpeed

c. Simulink Support Package for Parrot Drones

Simulink Support Package for the PMD allows users to build and deploy flight control algorithms on the PMD. Algorithms are deployed wirelessly over Bluetooth and a Wi-Fi network. The provided algorithms are employed to access the "ultrasonic, accelerometer, gyroscope, and air pressure onboard sensors, as well as the downward facing camera." [11] The algorithms are adapted and used within the Simulink add-on tools to "model 6-DoF equations of motion and simulate aircraft behavior under various flight and environmental conditions." [11] To simulate a planned trajectory within a virtual environment the Simulink support package provides an arena, and a modeled drone linked to a Simulink template with the PMD basic coding.

2. OptiTrack

OptiTrack is a high-performance optical motion tracker that provides optical tracking through motion capture software using high-speed tracking cameras [22]. The system used for the flights is pictured in Figure 12 with six cameras in a netted area for safety.

Figure 12. OptiTrack System in the lab environment

a. Camera: Prime X 41

The six cameras in the OptiTrack system are the Prime X 41, presented in Figure 13, that tracks a 3D location marker within plus or minus 0.1 millimeters accuracy with a frame rate up to 250 FPS and 4.1 MP resolution. 3D data is processed in real time resulting in live 3D data for the majority of all frames [23]. This style of camera is engineered to produce the most precise large-scale 3D measurements. It has a low distortion lens and true 10-bit grayscale depth to reduce quantization noise and improve precision. This design allows for a crisp image with defined centroids to provide the best 3D accuracy [24].

Figure 13. Prime X 41 cameras

C. IMPLEMENTATION OF SYSTEM

The experimental data essential for this research required a specifically ordered loop of the described hardware and software packages working together to launch the PMD. The system is set up by connecting the OptiTrack system to the host computer via ethernet to ensure continuous data and a Wi-Fi dongle attached to the host computer to transmit the desired commands to the PMD.

With all the systems connected, the flow of information begins with initializing the drone navigation code written in MATLAB. MATLAB establishes communication with Motive directly through ethernet, as well as with the PMD through Wi-Fi. The OptiTrack system tracks the PMD and sends the actual position of the drone to MATLAB. Then MATLAB uses this information to compute motion commands for the PMD, which in turn is transmitted via the Wi-Fi dongle to the drone. This system control loop continuously repeats until the PMD has accomplished the desired waypoint goals defined in the MATLAB drone navigation code. The system control loop diagram is presented in Figure 14.

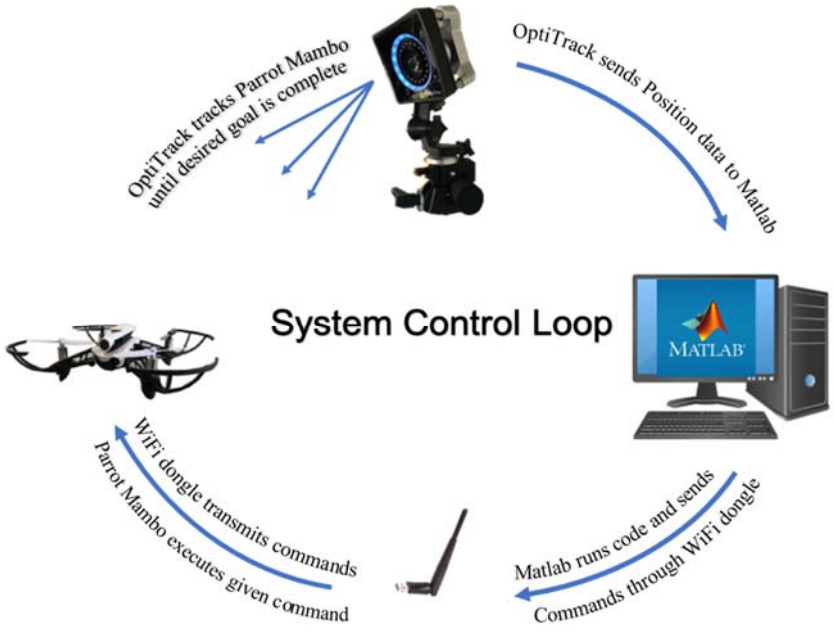

Figure 14. Control system loop of the Parrot Mambo Drone

IV. CONTROL DESIGN

It is a necessity to maintain control of a UAV for many reasons. The FAA has described control as the ability to establish, maintain or alter the attitude of an airplane in regard to its flight path [25]. UAVs have less autonomous control inputs than degrees of freedom, which creates a control problem when trying to maintain control of all 6-DoF. This creates a design opportunity to be incorporated to control the axes, along with, yaw, pitch, and roll.

The basic block diagram of the required inputs and desired outputs that a controller will need to properly control a UAV is illustrated in Figure 15. The objective of this chapter is to develop the basic equations for a PID control law, briefly introduce other control solutions, and introduce the control logic that is used for the simulated and experimental flights for this research.

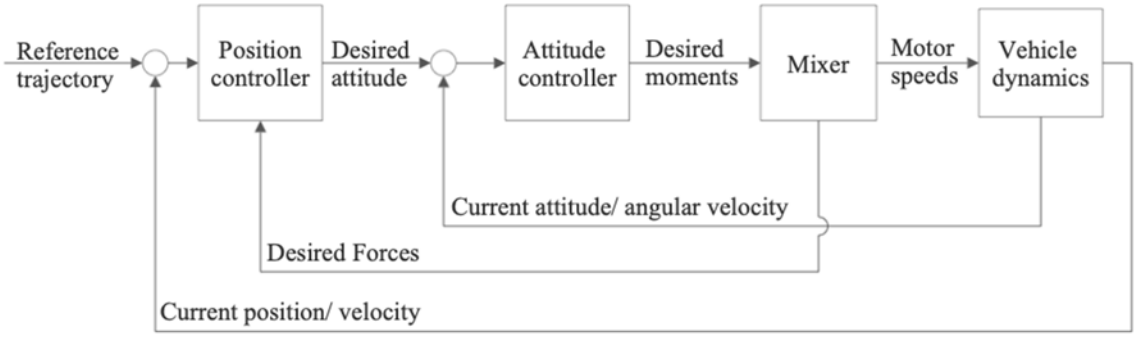

Figure 15. Quadrotor control system design. Adapted from [26].

A. PID EQUATIONS

In this section, an overview of the PID control algorithm will be presented. The main objective is to formalize a design for the PID controller to stabilize the flight of the PMD. The control input u used to maintain stability of the PMD, presented in [27], is designed as

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \dot{e}(t). \quad (23)$$

The gain K values are associated to their respective control mechanism: proportional gain K_p, integral gain K_i, and derivation gain K_d. The error $e(t)$ is formulated in [27] by

$$e(t) = s_p - p_v(t). \tag{24}$$

The desired position s_p is subtracted from the measured process variable p_v to provide a correction value to apply to the PID. It is required that the PID controller maintain stability in yaw, pitch, and roll while simultaneously controlling the error.

Efficient tracking of the PMD allows for a smooth and stable flight path. In Figure 16, a cubed space in the inertial frame that tracks the instantaneous position of the PMD through every adjustment is presented.

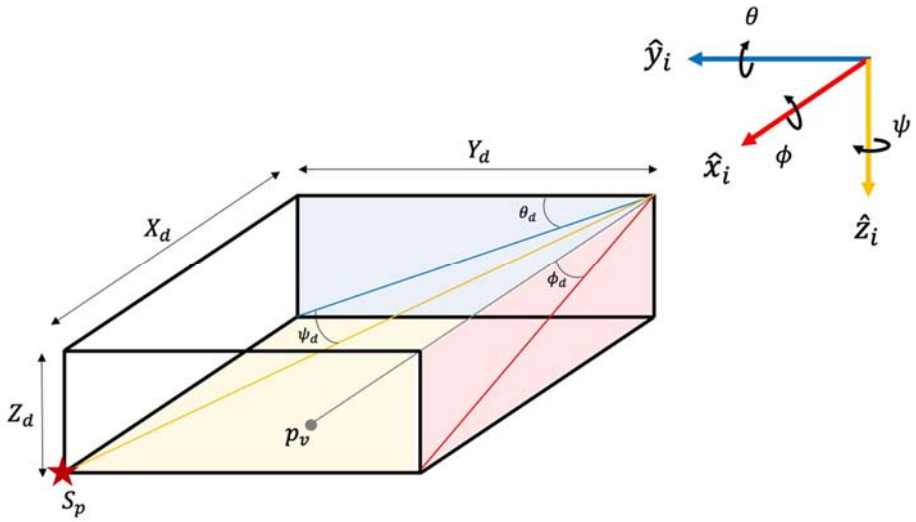

Figure 16. The PMD rotational angles description. Adapted from [27].

Each adjustment brings the PMD closer to the desired position. Achieving the desired position is accomplished by using the desired coordinate positions, X_d, Y_d, and Z_d, and the orientation angles, ψ_d, θ_d, and ϕ_d. To formulate the desired angles, as done in [27], the use of geometry depicted in Figure 16 is used with

$$\phi_d = \tan^{-1}\left(Z_d / Y_d\right), \tag{25}$$

$$\theta_d = \sin^{-1}\left(X_d \Big/ \sqrt{Z_d^2 + X_d^2}\right), \tag{26}$$

and

$$\psi_d = \cos^{-1}\left(Y_d \Big/ \sqrt{X_d^2 + Y_d^2 + Z_d^2}\right). \tag{27}$$

There are six control inputs that also must be formulated. They are described in [27] by

$$u_x = K_p\left(X_d - X\right) + K_i \int_0^t \left(X_d - X\right) dt + K_d \frac{d\left(X_d - X\right)}{dt}, \tag{28}$$

$$u_y = K_p\left(Y_d - Y\right) + K_i \int_0^t \left(Y_d - Y\right) dt + K_d \frac{d\left(Y_d - Y\right)}{dt}, \tag{29}$$

$$u_z = K_p\left(Z_d - Z\right) + K_i \int_0^t \left(Z_d - Z\right) dt + K_d \frac{d\left(Z_d - Z\right)}{dt}, \tag{30}$$

$$u_\phi = K_{p\phi}\left(\phi_d - \phi\right) + K_{i\phi} \int_0^t \left(\phi_d - \phi\right) dt + K_{d\phi} \frac{d\left(\phi_d - \phi\right)}{dt}, \tag{31}$$

$$u_\theta = K_{p\theta}\left(\theta_d - \theta\right) + K_{i\theta} \int_0^t \left(\theta_d - \theta\right) dt + K_{d\theta} \frac{d\left(\theta_d - \theta\right)}{dt}, \tag{32}$$

and

$$u_\psi = K_{p\psi}\left(\psi_d - \psi\right) + K_{i\psi} \int_0^t \left(\psi_d - \psi\right) dt + K_{d\psi} \frac{d\left(\psi_d - \psi\right)}{dt}. \tag{33}$$

The gains associated with each of the angle control inputs differ based on the stability criteria for each motion. The position is formulated with (29) and (30) to begin to orient the PMD. With (31) the altitude is achieved by implementing the same gain

values used for the position. When the coordinate positions are obtained, (32), (33), and (34) are commanded to the PMD with individual gain values for each specific orientation angle. After several iterations of these specific control inputs, the PMD arrives to the desired position.

B. OTHER CONTROL FORMS

For this research, only PID control is explored. There have been many studies conducted working on a solution for a perfect control design for a UAV. A few methods not explored in this research include:

- Model reference adaptive control is used by a controller that has varying system parameters and must compare the referenced value to that of what the controller outputs. This response creates various changes in the system from that response as discussed in [28].

- In [28], adaptive control is analyzed to explain how a system can modify its operations with unknown parameters to achieve a preferred performance design.

- Sliding mode controller is a nonlinear robust controller designed for complex nonlinear dynamic plants operating under unknown conditions [29].

- A dynamic system defined by a linear differential equation that is designed at minimal costs is a Linear Quadratic Regulator. The gain value is a given value and the permutation matrix is solved with the Riccati differential equation and the cost function is defined in a quadratic function explained in [7].

- Kalman filter, also referred to as linear quadratic estimation, produces estimates of the current state variables and the uncertainties connected to them. Then a new set of measurements is observed, and the system

updates the estimates using a weighted average. This system is recursive and can run in real time not requiring past information [28].

C. CONTROL LOGIC IMPLEMENTED

For this research, a simulated control system is analyzed, as well as an experimental control system. There are two control systems that are used to stabilize and control the PMD flight: an onboard controller designed by the manufacturer of the PMD and a supervisory controller designed for the use in this research.

1. The PMD Control Operation

The programmed controller is developed in Simulink from a model designed by Parrot. The full model details are explained in Chapter V. The flight control system will be explained in this section to analyze the control effects that are implemented. The model is designed with six different sets of PID controls. Each PID is designed with a specific gain value for its control requirement.

Shown in Figure 17 is the block diagram for the process that takes place for each flight adjustment the PMD makes. The system has two control loops, an outer loop and an inner loop, that feed continuously into each other. The system inputs are the position reference, estimated yaw, yaw reference, and the altitude reference. The state estimator for the Simulink model is broken into several filter blocks. The use of a complementary filter and Kalman filter are introduced to calculate the altitude of the system.

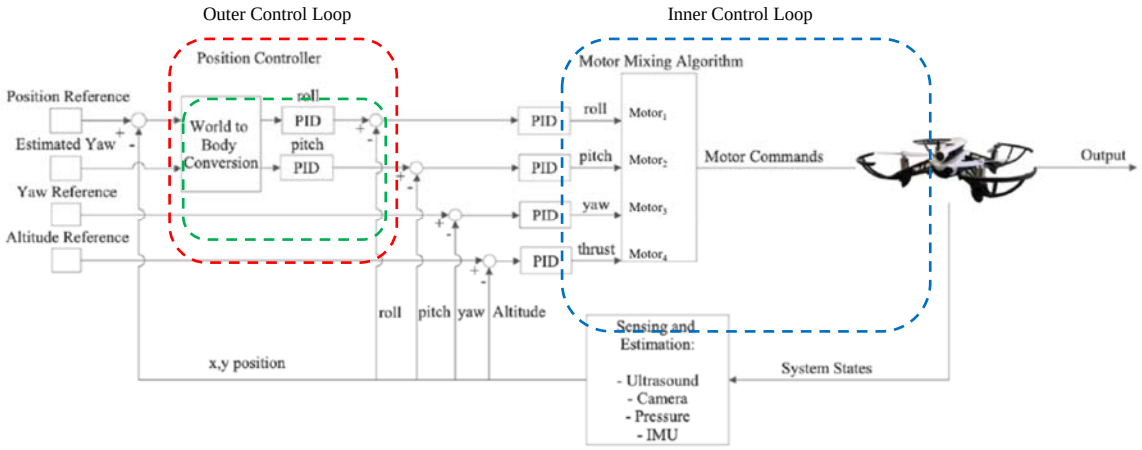

Figure 17. Simulink block diagram control logic for the PMD

The outer loop begins by converting the inputs of the position and yaw references from the world frame, inertial frame for this research, to the body frame. This process allows for the Simulink model to track where the drone is in comparison to where it is desired to be. The outer control loop ensures that the position of the PMD is within the desired tolerances. Once the PMD is in the coordinate positions, the system states are introduced as feedback into the inner control loop.

This section of the block diagram feeds from four PID controllers for roll, pitch, yaw, and thrust. These control inputs are sent to the motors of the PMD to create the motor commands. The commands produce an output from the PMD resulting in flight adjustments in all 6-DoF. These control loops continue to adjust until the PMD is at the desired waypoint.

Table 2. Simulink model gain values for six PID controllers

Control Input	Proportional Gain (K_p)	Integral Gain (K_i)	Derivative Gain (K_d)
u_x	-0.124	0.01	0.1
u_y	0.124	0.01	-0.1
u_z	0.8	0.24	0.5
u_ϕ	0.01	0.01	0.002
u_θ	0.013	0.01	0.003
u_ψ	0.004	0.01	0.3*0.004

The gain values for the Simulink model are originally defined by the manufacturer. These values are adjusted through a series of trial and error of Simulink runs to find the best set of gain values to complete a stable flight. The values in Table 2 represent the gain inputs for each of the PID controllers utilized in the Simulink modeled system.

2. Supervisory Control for the PMD

For the experimental flights, a supervisory controller is employed. This controller is designed to increase the performance of the PMD to reach the desired destination. The MATLAB code created has four PD controllers devised for the roll, pitch, yaw, and thrust.

The control system design is depicted in block diagram format in Figure 18. This system follows the same control logic as the Simulink model. The inputs are the position reference, yaw reference, and altitude reference. These values are tracked by the OptiTrack system and sent to MATLAB through Motive and then used to calculate the error in the system. With the coordinate positions defined, the position controller, inside of the supervisory controller, converts the world frame to the body frame before applying the PD control commands to the system. These values are added to the system state estimates to be feed into the motor mixing algorithm. This algorithm uses the four PD controllers to make the roll, pitch, yaw, and thrust motor commands that will be sent to

the PMD controller explained in the PMD operation. After going through both sets of controllers, the motor commands are sent to the PMD to complete that set of movements. These iterations continue until the PMD has reached the desired waypoint.

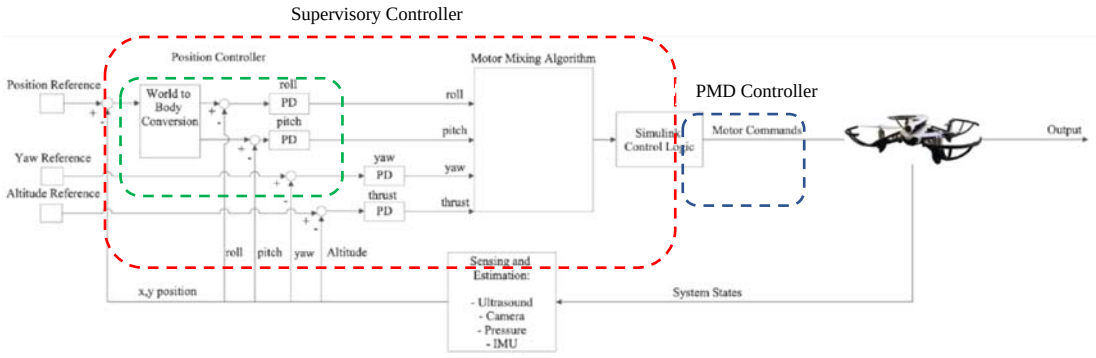

Figure 18. Experimental flight block diagram for the PMD

The gain values employed in the MATLAB code are obtained through multiple trial and error runs to establish the best set of gains for the system. Table 3 has the final values used for the experimental flight.

Table 3. Experimental model gain values of four PD controllers

Control Input	Proportional Gain (K_p)	Derivative Gain (K_d)
u_x	-0.2	0.1
u_y	0.2	0.1
u_z	0.9	0.1
u_ϕ	-0.086	-0.1
u_θ	0.086	0.1
u_ψ	0.9	0.3*0.6

3. Design Parameters

To validate the control procedures put in place for this research, specific tolerances are established for the positions and input command values listed in Table 4. The focus for this research is on position of the three-axes overall error.

Table 4. Design parameter for the PMD simulated and experimental flights

Parameter	Allowed Tolerance
XY Position	0.2 meters
Yaw	3 degrees
Turn Duration	0.3 seconds
Vertical Duration	0.8 seconds
Yaw Command Input	180 degrees
Roll Command Input	25 degrees
Altitude Command Input	±2 meters
Pitch Command Input	25 degrees

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULINK SIMULATION OF THE PMD

The Simulink Support Package for Parrot Minidrones (SSPPM) is used to examine the performance of the PMD in a simulated environment. The SSPPM permits the building of flight control algorithms and the ability to deploy flight on the PMD wirelessly over Bluetooth [30]. For the purpose of this research, simulated flight is analyzed, and Bluetooth flight is not performed. The comprehensively modeled environment specific to the PMD contains a simulation model with the necessary sensor dynamics, nonlinear mathematical model, and integrated flight control system with tunable PID controllers. The focus in this chapter is on the flight control system that is manipulated with the design parameters desired for the experimental flight. The SSPPM modeled tree is described along with the 3D visualizer that is employed while the Simulink model runs. The desired waypoints (WP) are described along with the simulated results.

A. SIMULINK MODEL FOR THE PMD

The SSPPM is equipped with multiple modeling reference examples. For this research, the Simulink model used is the *parrotMinidroneWaypointFollower* [36]. This command, when entered into MATLAB, launches the project folder with the initial parameters and models to execute a simulated flight path. This program can either simulate the desired flight path, with the 3D visualizer, for the PMD, or deploy the Simulink designed code on to the PMD via Bluetooth to execute the flight path. For this research, the example project starts the flight of the PMD and follows the pre-configured set of WPs for the flight path. The simulation will execute a simulated flight path to be loaded onto the PMD to simulate the flight or deploy using a Bluetooth connection to complete the flight path.

The original flight model of the PMD [36], as shown in Figure 19, contains subsystem component blocks for flight commands, flight control system, simulation model, and a flight visualization. The flight control system blocks are the focus of control adjustments.

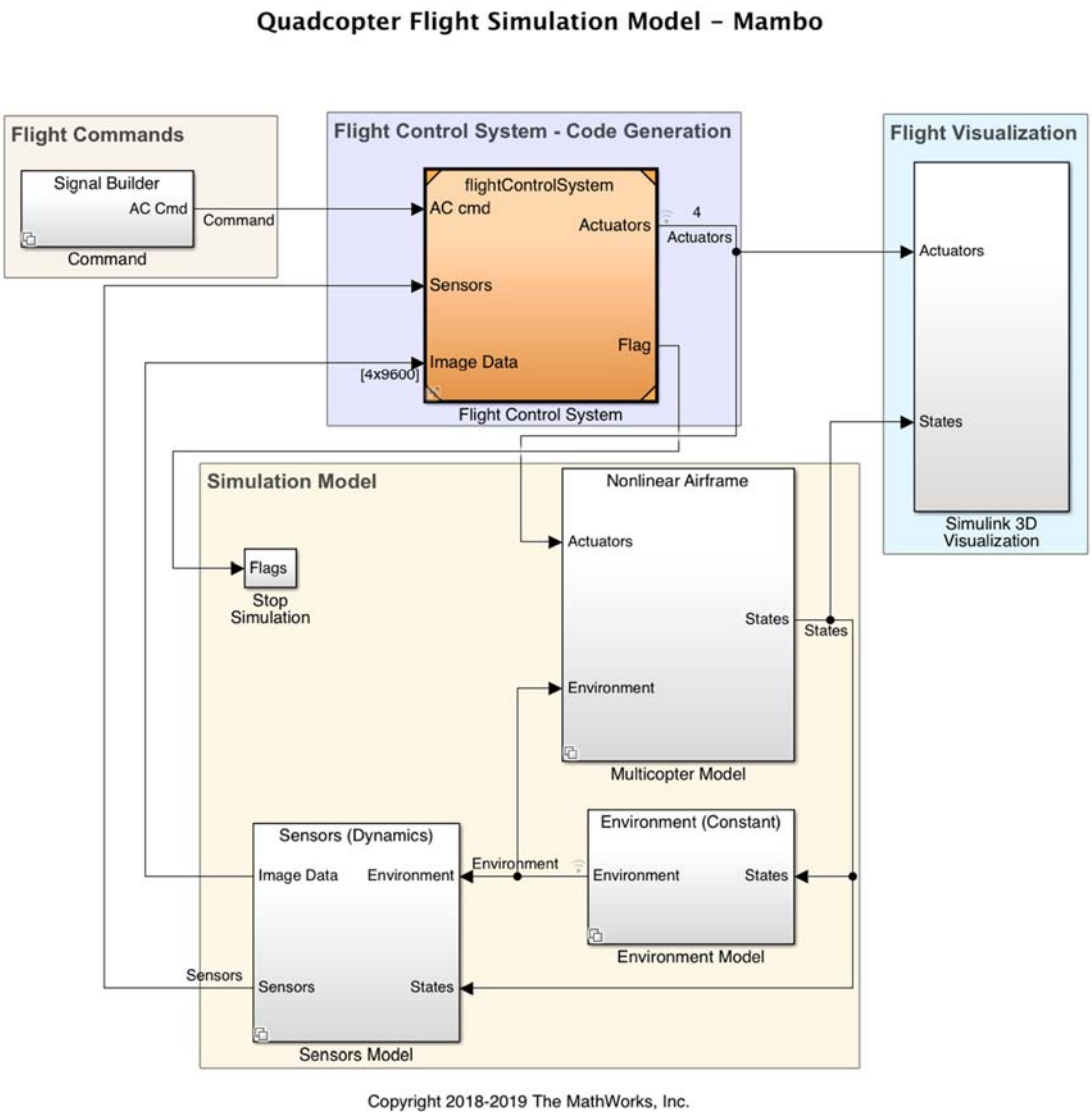

Figure 19. Quadcopter flight simulation model. Source: [30].

There are two main subsystem blocks that make the flight control system for the PMD, as shown in Figure 20. There are no modifications made to the image processing system, and the image processing system is not discussed because it is not used in this research. The flight control system is manipulated to achieve the required responses and to emulate the desired flight path implemented for the experimental flights.

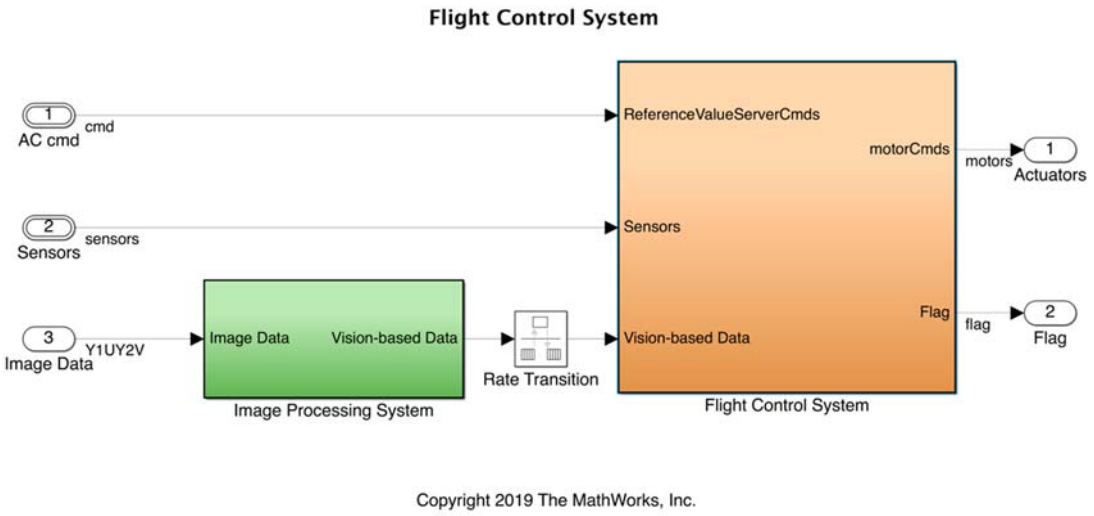

Figure 20. SSPPM flight control system design. Source: [30].

1. Flight Control System

The flight control subsystem, as shown in Figure 21, controls path planning and the type of controller that is applied. In an ideal situation, position and control would be controlled simultaneously. To accomplish this the state estimator, the path planning, and the controller blocks are modified to obtain the desired output of the PMD.

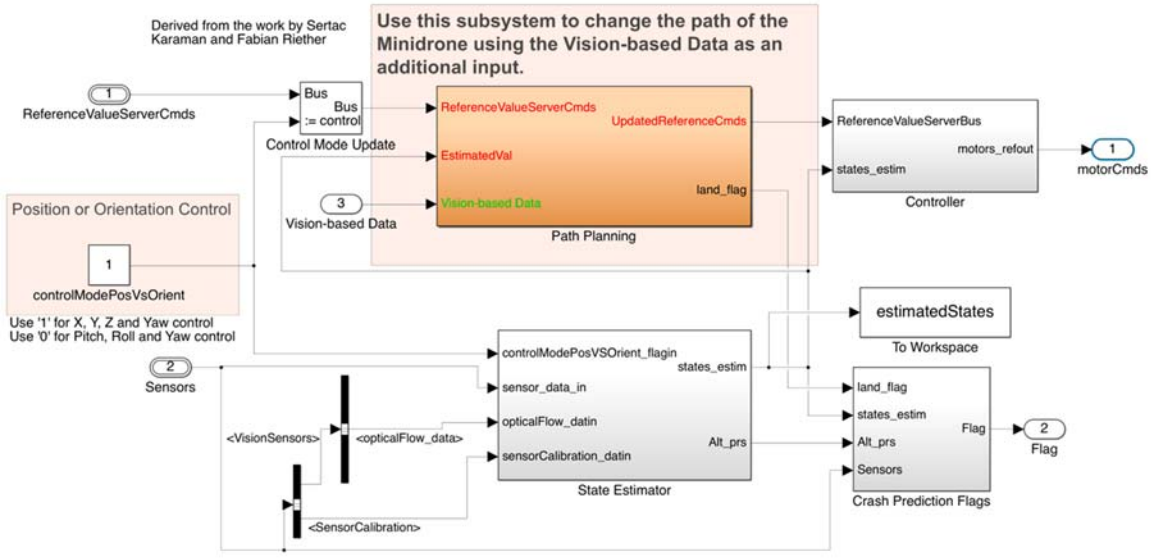

Figure 21. SSPPM original internal components for the flight control subsystem for the PMD. Source: [30].

a. State Estimator

A twelve-dimensional state vector for the PMD is created within the state estimator subsystem, Figure 22, consisting of position, orientation, velocity , and angular velocity $\begin{pmatrix} x & y & z & \phi & \theta & \psi & v_x & v_y & v_z & p & q & r \end{pmatrix}$. The state estimator block uses the four PMD sensors to collect data and compile this information.

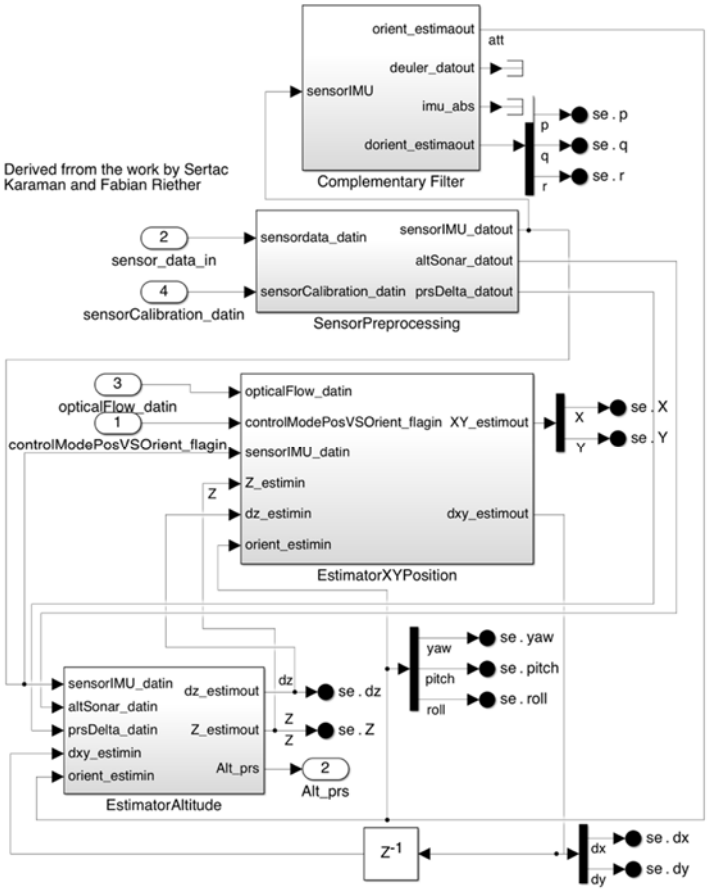

Figure 22. SSPPM original internal components for the flight control state estimator subsystem for the PMD. Source: [30].

The state estimator, Figure 23, is comprised of two subsystem blocks, the velocity estimator, and the XY position estimator. For this research, it is required to enhance the XY position subsystem to achieve the desired output parameters listed in Table 4.

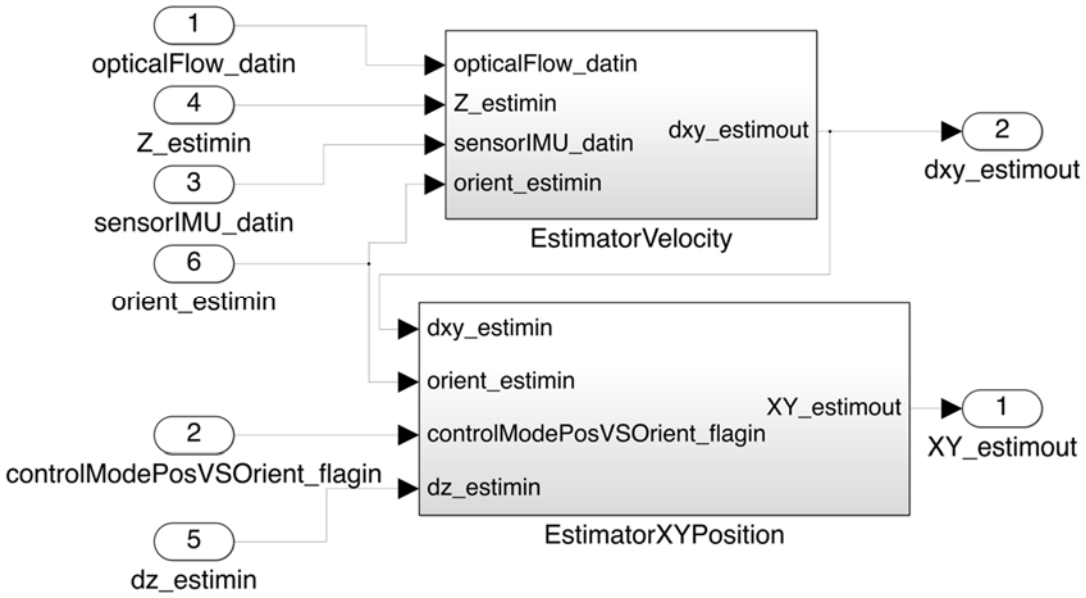

Figure 23. SSPPM subsystem of the original upper level estimator for XY position of the PMD. Source: [30].

In Figure 24 the original lower level subsystem of the XY position estimator is seen. During simulated flight trials the system could not perform within the desired tolerances. To adjust the output of the system, a simple integrator block and Kalman filter are introduced to use the previous estimated positions and the integrated velocity to estimate the current position.

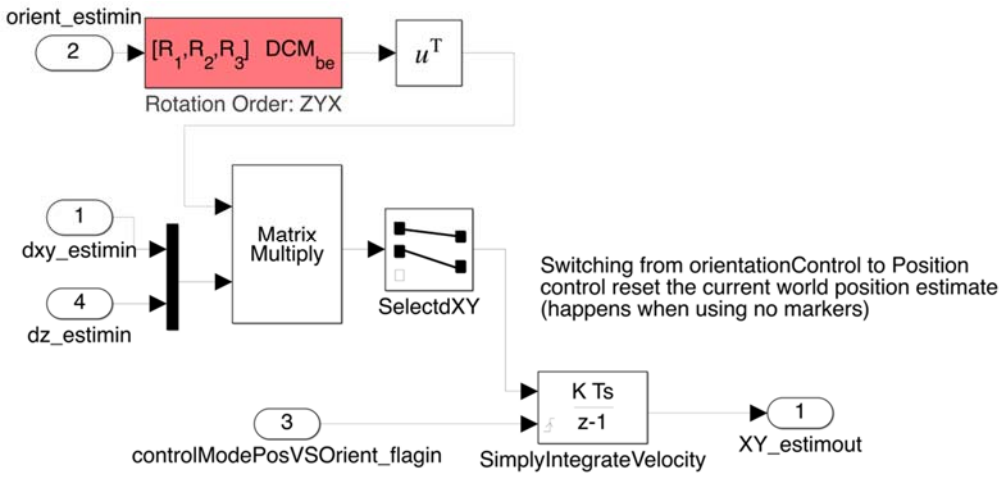

Figure 24. SSPPM of original lower level estimator for XY position of the PMD. Source: [30].

To ensure the simulation performed within the desired parameters, the Simulink model in Figure 25 is created from [10], [31], and [32]. The Kalman filter block uses the position data from the Simulink reference port to obtain the measured position for the Kalman filter. The simulation estimated velocity is used as the command for the position estimation. With this data, three states are created in which the Simulink model will operate: initial, lost, and confirmed.

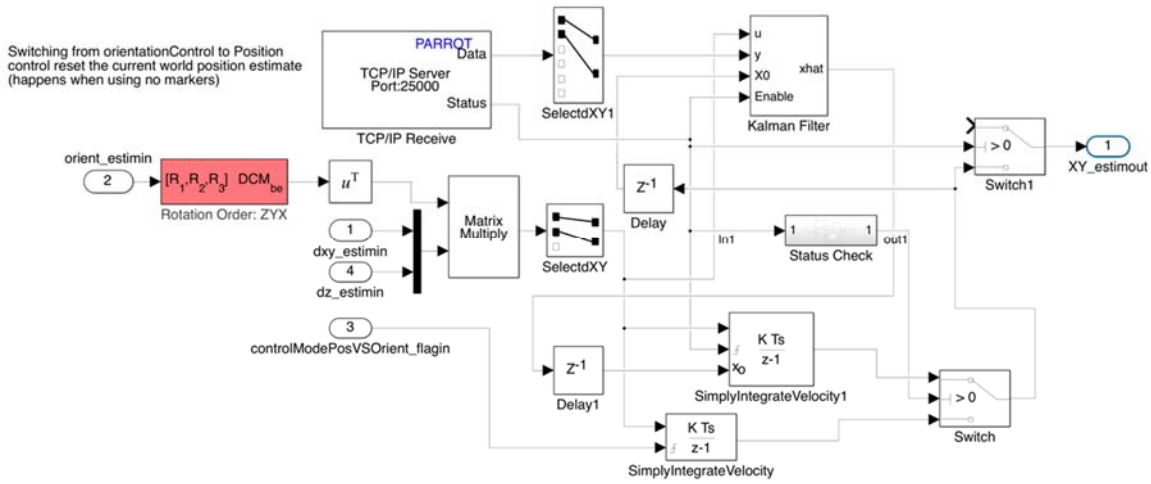

Figure 25. SSPPM of modified lower level estimator for XY position of the PMD. Adapted from [30].

The initial state is designed to ensure the PMD does not begin to execute commands before communication with the reference host is established. For the simulation, a TCP/IP Receive block is employed to act as an outside reference source. While the PMD is in this state, the system assumes the PMD is at the initial XY state of [0;0] and integrates the velocity in the X and Y positions to establish the current position. After the state is determined, the system moves to the lost or confirmed state [32][10].

The lost state is enabled when the PMD has not had a status update for the X and Y positions. When this is triggered, the previous time step estimate of the previous position, from the Kalman filter block, is used and the system integrates the X and Y velocity to obtain the current estimate position in the X and Y. From the TCP/IP Receive block, the initial conditions are reset to the falling edge of the status signal.

The final state is the confirmed state that engages the Kalman filter block. The Kalman filter relies on noise associated with the plant and measurements [10]. From the original SSPPM model, the noise values are estimated. The model is designed to reduce the size of the uncertainty bubble [10] with each estimation. When the system transitions out of the confirmed state and is required to enter the lost or initial state, the uncertainty bubble returns to the original initial or lost state size.

To accomplish reducing the size of the uncertainty bubble, the system uses the received position as the measured position and the X and Y velocity as the position commands. The initial conditions, for this status, are the estimated position from the previous time step from the initial or lost state.

b. Desired Path Design

Path planning, Figure 26, in the SSPPM subsystem contains two block designs to ensure proper flight of the PMD. This first design block is the logic to land the PMD, which will be triggered if a warning flag is activated, or if there is less than five seconds remaining of the flight. The second section is the waypoint follower system and is manipulated for this research [30].

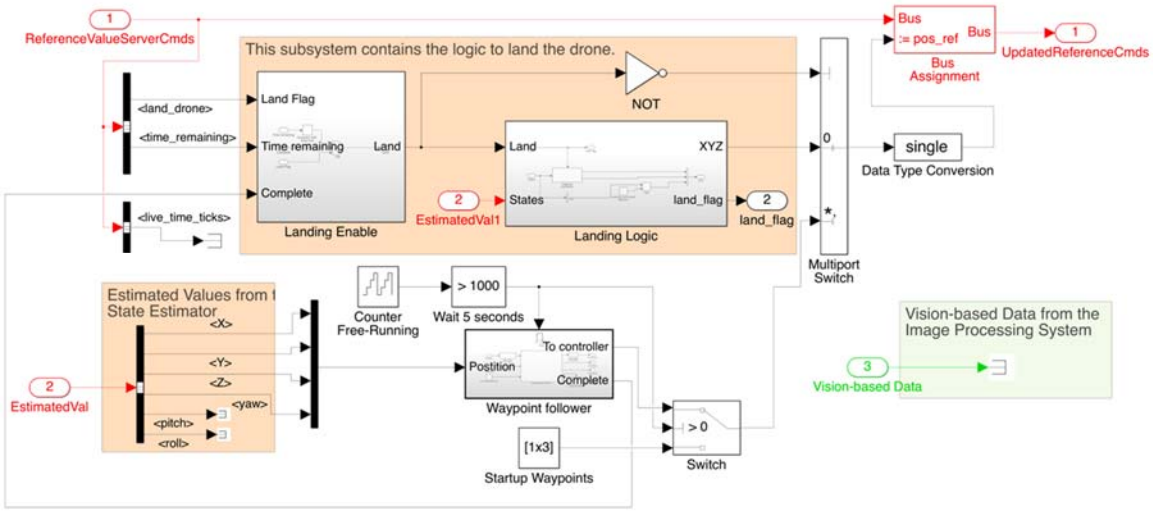

Figure 26. SSPPM subsystem of the original upper level path planning of the PMD. Source: [30].

Given the PMD position, WPs, and look ahead distance, the waypoint follower, from the Robotics System Toolbox UAV library, calculates the desired yaw, heading, and lookahead point [33], depicted in Figure 27. The WP box is filled in with the desired WPs established for this research, given by

$$WP = \begin{bmatrix} 0 & 1.5 & -1 & -1.5 & 0.5 & 1 & 0 \\ 0 & 1.5 & 1 & -1.5 & -0.5 & 1 & 0 \\ -0.8 & -0.8 & -1.2 & -0.8 & -1.2 & -0.8 & -1.2 \end{bmatrix}. \tag{34}$$

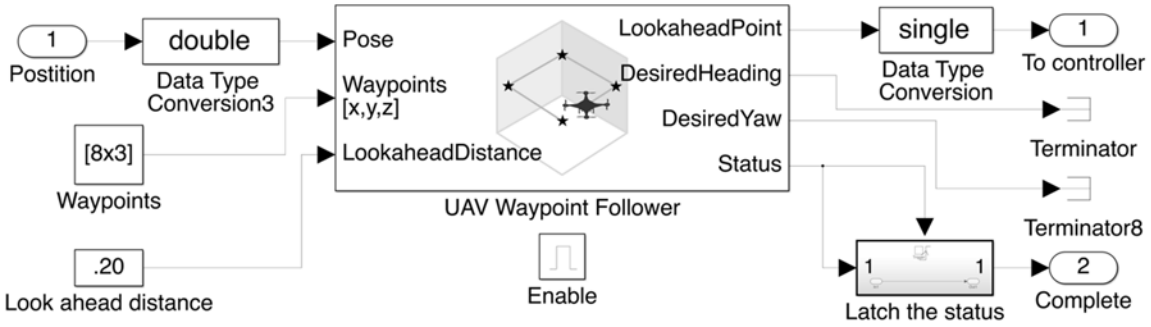

Figure 27. SSPPM of modified lower level waypoint follower of the PMD. Adapted from [30].

The chosen WPs, Figure 28 and (34), are designed to test all motions and rotations of flight for the PMD. Each WP has a different desired position, yaw, pitch, and roll requirement. This design ensures that all control commands implemented can be used for any flight pattern desired. Figure 28 has the order and direction of flight for the desired WP goal locations.

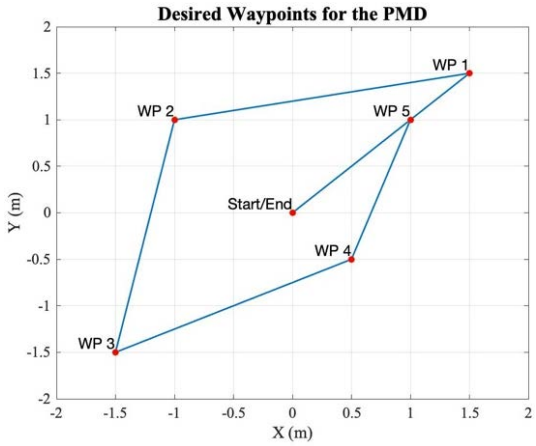

Figure 28. Established WP for the PMD flight path

c. Controller

The focus of this research is on the control implementation for the PMD. In the SSPPM, the controller subsystem, Figure 29, is used without additional modifications. However, the internal blocks are adjusted so the gain values produce a stable efficient flight path for the simulation of the PMD [33]. The final values used in the subsystems are reflected in Table 2.

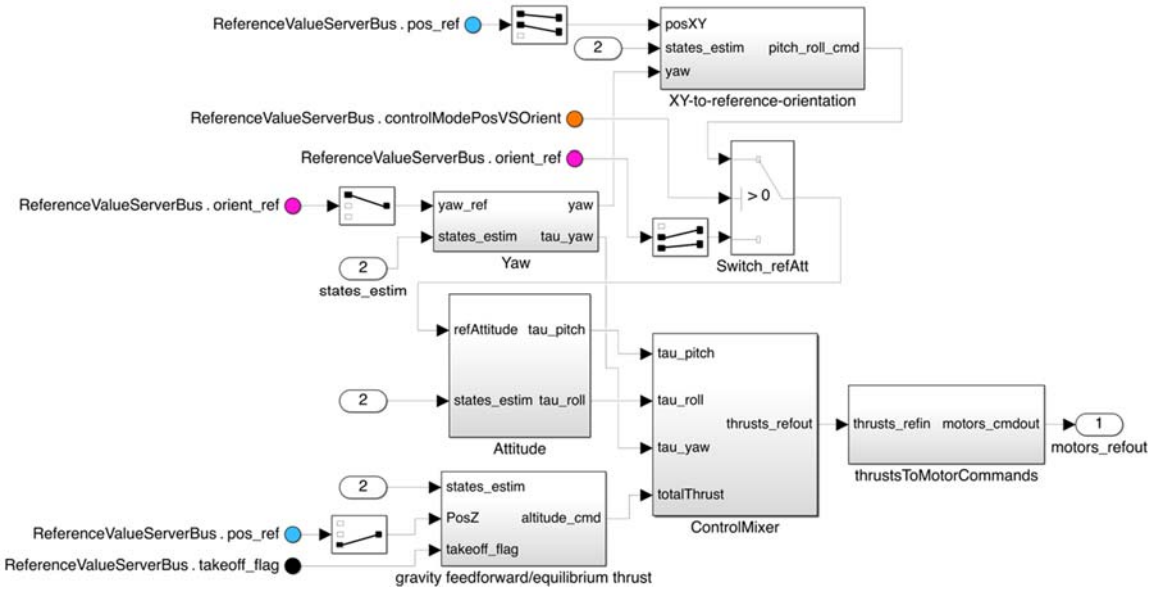

Figure 29. SSPPM subsystem of the original upper level path planning of the PMD. Source: [30].

2. 3D Visualizer

The SSPPM for the PMD is equipped to be viewed from three different perspectives while simulating the desired flight path. An isometric, chase, and quadrotor camera view are available in the Simulink 3D visualizer.

In Figure 30, each viewing angle depicted provides a specific orientation outcome. The isometric view is used to view the rotations of roll and yaw coupled together while the PMD is in flight. The chase view is designed to view the altitude and roll reactions. The final view is the quadcopter view that allows for a top-down tracking while the drone is in flight [33].

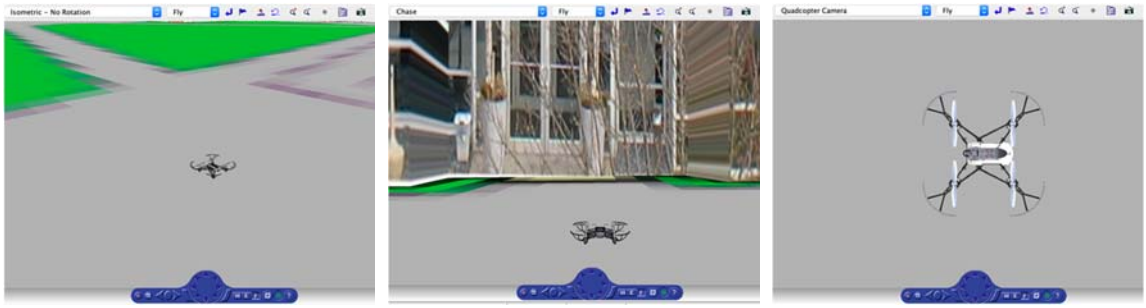

Figure 30. The PMD isometric, chase, and quadrotor camera views. Source: [30].

The quadrotor view in Figure 31 is used during simulated flight to depict how well the model is following both the desired WP and trajectory that are implemented. The red circles demonstrate the 0.2-meter transition radius tolerance that is allowed for the research.

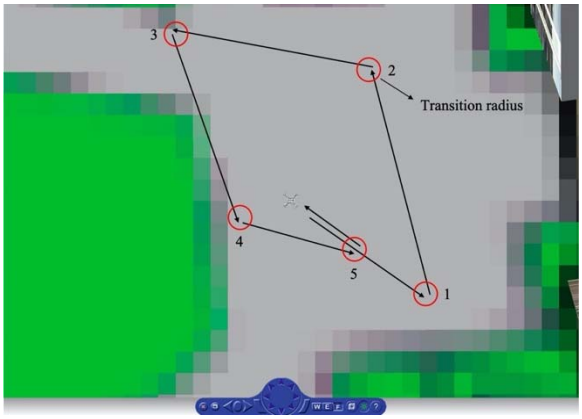

Figure 31. The PMD Simulink modeled waypoint desired path. Adapted from [33].

B. SIMULATION RESULTS

Work in [5] by Allen is done on a constructed simulation of a quadrotor; however, there is a difference between a mathematical model simulation and a physical construction of a modeled quadrotor. The significant difference between the two is the accuracy in estimations. In this chapter, the simulation results for the PMD system states in the SSPPM for the optimal control design are presented. The procedure for the Simulink model is explained at the beginning of this chapter. All the results are grouped by their respective 6-DoF state.

1. The PMD Simulated Flight Path Results

The simulated flight path of the PMD is presented in Figure 32. The PMD is flown to six desired WPs, depicted by blue circles, in a specific order and are tracked by the OptiTrack system to evaluate the accuracy of the PMD input commands. The three-dimensional view illustrates that the gain values chosen, Table 2, adequately commanded the PMD to the desired WPs. The grid spacing represents one meter, and the star is the start and finish of the simulation. During the simulation, the initial start of the PMD is at one meter. This is a setting within the SSPPM does not move vertically, but at an angle to achieve the first WP. This occurs due to the initial parameters designed in the SSPPM. The following WPs are achieved by obtaining the desired heading, yaw, and then moving to a calculated distance at a specific speed.

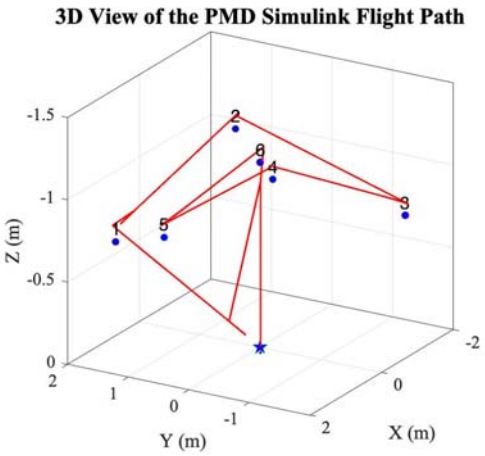

Figure 32. Flight path of the PMD during simulated flight

To better visualize the flight path taken by the simulated PMD, a plot of the PMD position in the X and Y axis is displayed in Figure 33. From this top-down view, the flight path indicates that the PMD went to the desired WP in the most efficient manner.

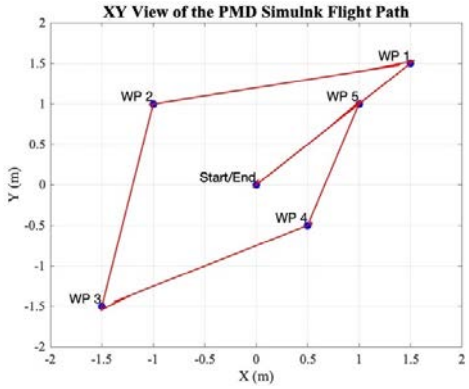

Figure 33. Top-down view of the PMD simulated flight path

2. The PMD Simulation Response and Error Values

In this section the individual response for each axis and range of motion is presented along with the errors associated with each axis. Displayed in Figure 34 is the X, Y, and Z axis response throughout the entire flight of the PMD. The plots are viewed with time versus the respective axis position in meters. The red line is the position that the PMD is in for the simulation. Each WP is marked with a blue dashed line, labeled

appropriately, at the point the PMD reaches the desired goal in the simulation. The solid blue lines depict the amount of time the PMD is in transition from one WP to the next.

For the simulated flight, the plots arrive at the desired locations well within the limit of 0.2 meters, set as the main parameter for this research. In Figure 34 and Figure 35, the axis and rotations are broken into sperate graphs to view the response for each one individually. It is observed in all plots that from the start to WP 3 there is a slight disturbance and a delay to flight. The delay occurs from the SSPPM design that is adapted to have the simulated model begin one meter off the ground when initialized. For this research, all flight paths begin and end with all states equaling zero. This forced the simulation to move from one meter off the ground down to the ground to ensure all states were at zero. The disturbance is observed more prominently at the beginning of the simulation due to the model adapting to the correct position with the Kalman filter that is implemented, along with the preprogrammed control systems designed within the SSPPM.

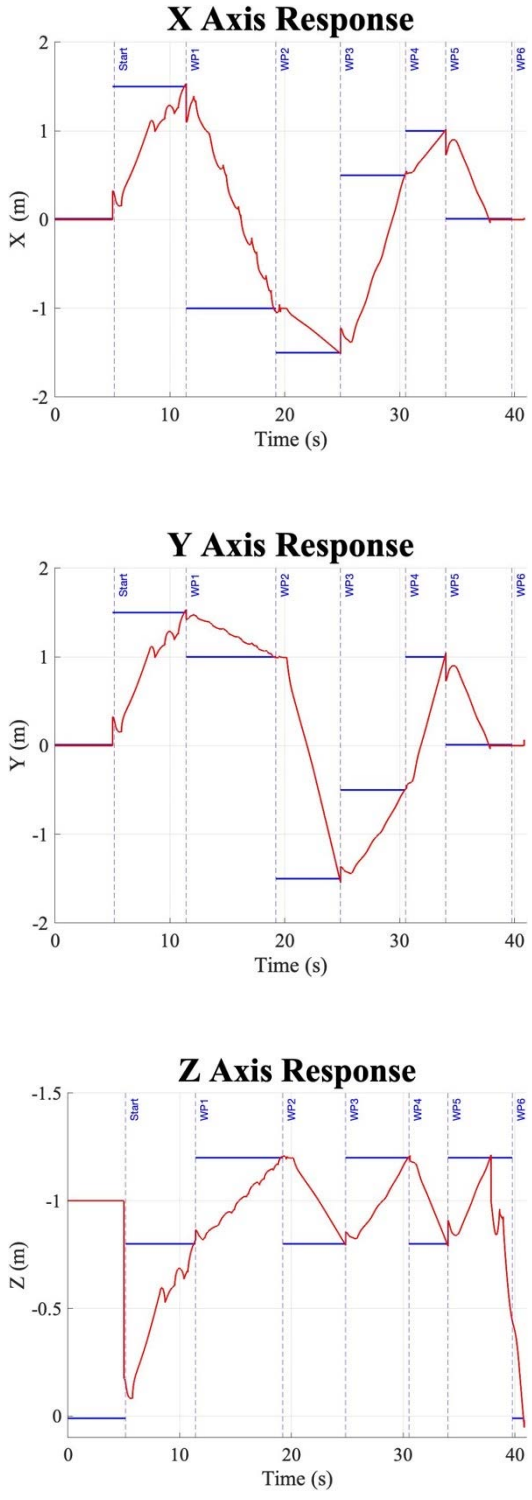

Figure 34. Three axes (X Y Z) distance response for the PMD

The command inputs for yaw, pitch, and roll for the PMD, depicted in Figure 35, are evaluated in degrees versus the total time in seconds taken for the simulated flight. Each WP is marked with a blue dashed line at the moment the system has reached the desired WP. The control law limited the response movement to nothing greater than 0.24 degrees.

It is observed that there is a spike in input at the end of every WP destination goal. This is attributed to the PMD making the required adjustments to orient to where the next WP is. WPs 3 and 5 have a small command input due to the simulated PMD already having the desired heading and not requiring much thrust to move to the next WP. The plots show very little noise after the initial command inputs to the motors. The simulated environment did not account for an outside factor such as weather. This allowed for an unhindered analysis of the gain values, listed in Table 2, that could be applied to the PMD to achieve optimal control.

For this simulation, there are not any additions to the control measures besides what are built onboard the PMD. Each value is adjusted to ensure the simulated PMD would reach each WP with the desired tolerances for the research.

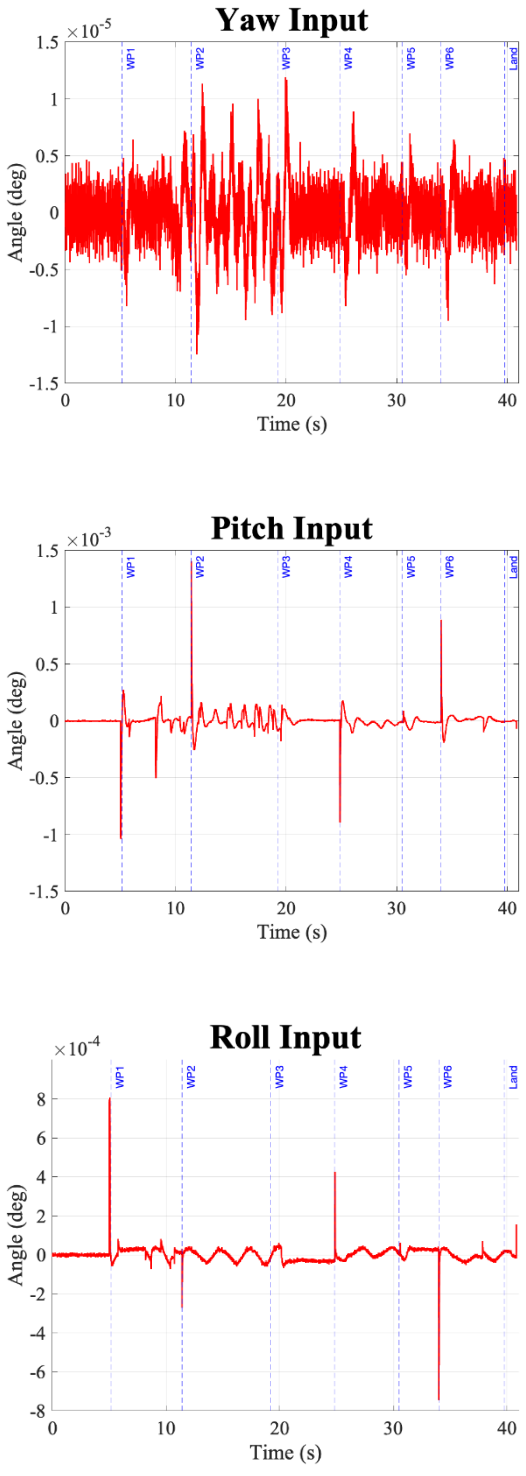

Figure 35. Three rotations (Y P R) for the PMD inputs

The distance traveled and command input values are important to analyzing the effectiveness of the control design for the PMD, but the analysis of the error values allows for the ability to fine tune the PMD motions. The data in Figure 36 are the final error values throughout the PMD flight to the specified six WPs. The plots are grouped by position and angle, measured in meters and degrees respectively, versus the time in seconds the PMD required to complete the simulated flight. All errors for the PMD simulated flight stay within the 0.2 meters and 0.1 degrees required parameters listed in Table 4.

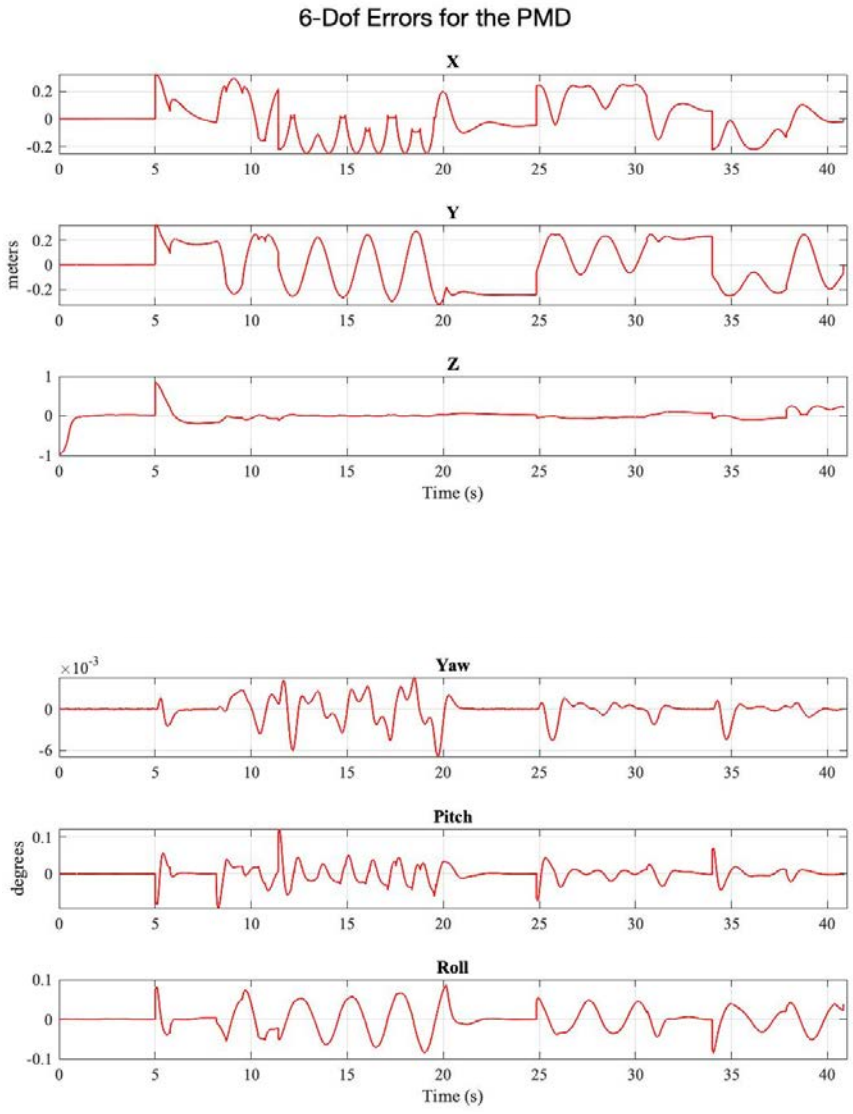

Figure 36. Error for all 6-DoF for the simulated PMD

Table 5 is compiled with the final percentage error value at the end of each WP flight iteration. The PMD is restricted to the values in Table 4 before it can proceed to the next WP goal. Each control parameter for the simulated flight of the PMD stayed well within the required parameters to meet the goals of the research.

To reach the initial WP, there is a large percentage error of 36.9% in the Z axis. This error is due to the initial programming in the SSPPM starting at one meter and having to move down to zero meters. This time and movement is calculated against the PMD error for WP 1. The following WPs are consistent and stay within a 5% average error for the four parameters that are evaluated to calculate when the PMD has reached the desired WP within the design standards.

Table 5. Percent error at the conclusion of each WP for the simulated PMD flight

Position	**WP 1**	**WP 2**	**WP 3**	**WP 4**	**WP 5**	**WP 6**
X	0.0570	1.5528	5.8818	9.3250	1.3890	0.0005
Y	0.0077	2.1878	4.2353	1.1141	5.9354	0.0008
Z	36.9985	5.7280	1.5203	1.7386	3.7305	5.4561
Yaw	0.0259	1.5774	6.7472	0.0429	3.7556	0.3043
Average Percentage Error	9.2723	2.7615	4.5961	3.0551	3.7026	0.0003

VI. EXPERIMENTAL RESULTS

In this chapter, results from the experimental flights are presented along with a comparison of the simulated flights to the experimental. The MATLAB code that implemented the supervisory control law for the PMD is described in detail. The experimental results are presented, with results from a trial with constant altitude, as well as a trial accounting for all 6-DoF motions.

A. MATLAB CODE

To simulate an autonomous drone, a control law within MATLAB is coded to optimize the flight of the PMD. This control design acts as a supervisory control, described in Chapter IV, to establish a robust control for the PMD structure. This code implemented tolerances, Table 4, and specified the optimal gains, Table 2 and Table 3, that the PMD follows to achieve each desired WP in the most efficient path.

The code establishes a global variable for the drone control loop operator to ensure the PMD completes all desired WP destinations. The MATLAB code also creates a control handle that launches when the script begins, in order to run to be able to abort a flight in case of an emergency or issue with the launch. The final initialization of the script is to establish a connection with OptiTrack, as well as the PMD.

The MATLAB code Drone Navigation, Appendix A, is designed with a proportional and derivative control law (PD). From the equations in Chapter II, PD gains for yaw, pitch, roll, and altitude are determined, as listed in Table 3. To implement the gain values, the commands for each 6-DoF are needed. This is accomplished first by determining the errors for each state. These values are assessed with the OptiTrack system and sent to MATLAB to compare to the desired WP value that the PMD is meant to achieve. The commands that are calculated are limited by the PMD capabilities. For the roll and pitch, the PMD could not exceed 25 degrees, and the altitude command could not exceed ± 2meters per second. To move the PMD, the command ‘move’ is used. It is established that when using the specific commands, the PMD performance is significantly degraded, found in Table 1. Applying calculated values for each 6-DoF

movement or rotation and specifying the turn duration with the rotation speed, vertical speed, pitch, and roll individually resulted in a smooth and efficient flight pattern.

The Drone Navigation code is designed with the use of switch cases. These cases are a function in MATLAB that evaluates one of several groups of statements. This means that there are several different cases, and the code will run through all of them and evaluate which case is appropriate for the current situation. The code will move to the next case when the case expression is defined as being true. For this research, the following six switch cases are implemented: take off mode, yaw mode, position mode, vertical mode, hover mode, and land mode.

The take-off mode is used to ensure that the PMD is receiving and connected to the OptiTrack system before moving. There is a 0.1 second pause forced on the PMD before any flight begins. This duration is chosen due to it being the appropriate time step required to allow OptiTrack data to begin transmitting to MATLAB. While in this state, the MATLAB script will output ‘waiting for altitude’ to let the user know the PMD is connected and preparing for flight. Once the PMD had received the appropriate amount of data from MATLAB, the PMD is prompted to switch from take-off mode to yaw mode.

For this research, the yaw is set to always maintain a heading of zero degrees. This is used for consistency with the simulated model, as well as the two experimental trial flights. While in the yaw mode, the MATLAB code ensures that the yaw angle is within the required tolerance. The PMD will adjust as required, using the yaw command that is computed at the beginning of the script before moving to the position mode.

The position mode is the case that creates the movements required for the PMD to move from one WP to the next. This case controlled the roll and pitch. After each adjustment, a pause of 0.055 seconds is implemented. This is designed to ensure that the information coming from OptiTrack to MATLAB is being fed to the PMD in a timely manner. There is a slight delay with information being processed from OptiTrack to MATLAB and then to the PMD. Without a small delay after each movement, the PMD is prone to fly off track. With the small workspace, it is imperative to ensure the PMD does

not go more than 0.5 meters off the desired path. When the PMD reaches within 0.4 meters of the desired goal, a new set of commands is applied. It is observed that when the PMD is within this range of the desired WP, the computed commands do not provide enough thrust to adjust the PMD. This is solved by adding a move command that is twice as much as the calculated command, but half the turn duration. While in this state, the MATLAB code will output a message of, 'X Y error of (current error) m not within tolerance of (desired tolerance) m.' With this applied, the PMD is capable of maneuvering in and out of desired WPs with little issue. The final section to this case is to ensure the PMD is stable before proceeding to the next WP. This is accomplished by implementing an 'if' statement with a move command with a value of zero for the roll, pitch, and rotation speed. This state is held by a counter for two control loop cycles. Once the PMD is stable and achieves the X, Y, and yaw position of the WP, the case will switch to the vertical mode.

The vertical mode is implemented separately from the position. This design approach is used to complete a more accurate position placement. It is observed that when altitude and position commands are applied at the same time, the system becomes unstable. This is attributed to the lack of an integral controller, which is discussed in the conclusion in Chapter VII. This case is required to reach the desired altitude. When the altitude is at the desired WP, the system will hold in place with a counter for a count of five, as when in the position mode. Before the PMD can leave this state, it checks to ensure that it is within the position tolerances. If it is not, the PMD is prompted to switch to the position mode to acquire the desired position. Once the PMD achieves these cases it is prompted to enter the hover mode.

The hover mode is established to give a final check that the PMD is within the limits that the user has defined. The PMD will hover for a count of two hover iterations before moving to the next WP iteration. When this occurs, the PMD is commanded to go to the yaw mode switch case and run through the required cases. Once the PMD has completed all the desired WPs and is in the hover mode and within the required tolerances, the PMD switches to the land mode. The land mode takes the PMD out of the drone control loop and ends the flight by giving the PMD a land command.

B. EXPERIMENTAL RESULTS

To accurately analyze the control law that is implemented for the flight of the PMD, two experimental flights are used. Both maintained the same desired WPs and maintained the same gain values, listed in Table 3. The difference between the trials is one trial maintained a constant altitude of 0.8 meters. This altitude is chosen because it is the height that the PMD is preprogrammed to reach when it begins a flight. The simulated, experimental constant altitude, and full experimental flight trials compared to each other with two sets of WP values for each, is covered in this chapter.

1. Constant Altitude Flight for the PMD

In Figure 37, a three-dimensional diagram displays the flight path that the PMD performs. Each WP is marked with a blue circle and the corresponding number for the order that the flight occurs. The start and finish of the flight is marked with a blue star. The measurements for all flights are in meters for position and degrees for angular movement.

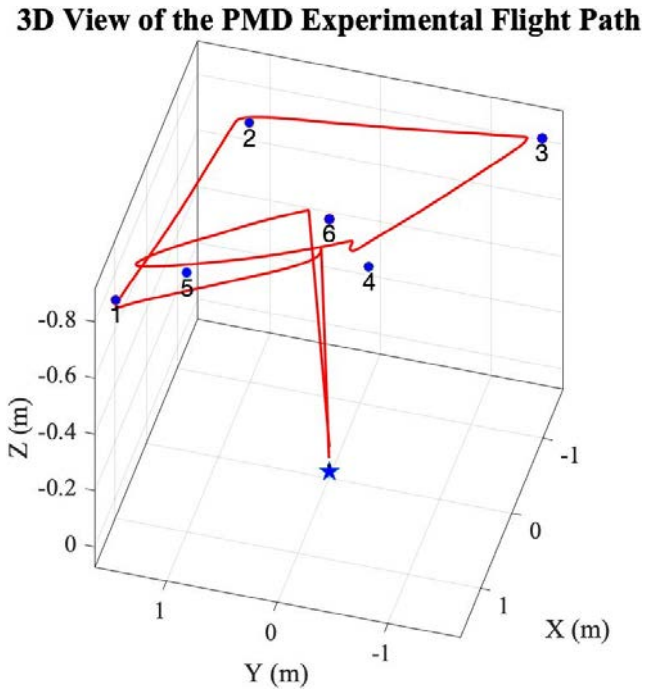

Figure 37. Flight path of the PMD with a constant altitude value

To better analyze the accuracy of the PMD in reaching the desired WPs, a top-down view of the X and Y axis is presented in Figure 38. The tolerance of ensuring a 0.2-meter error before moving to the next WP can be verified by this plot. The flight path has rounded corners in comparison to the simulated model in Figure 33. This can be attributed to the PMD still having momentum before adjusting to the next WP. In the simulated model the perfect conditions allowed for the PMD to be in a perfectly stable hover state before proceeding, giving the plot sharp corners from each WP.

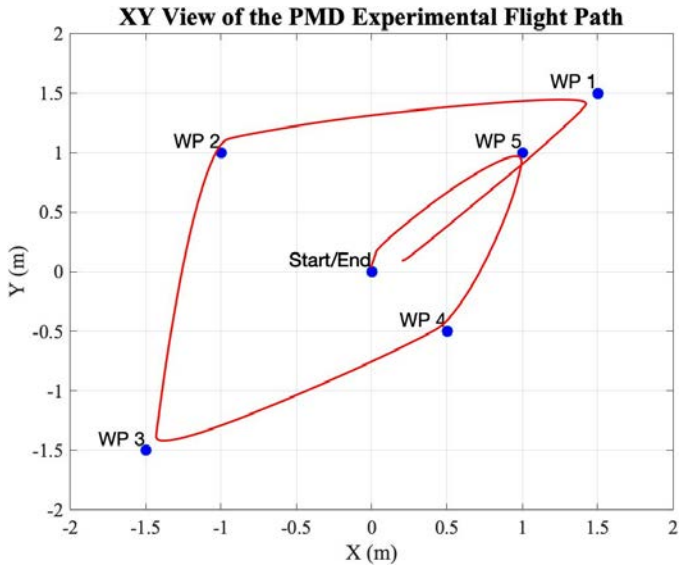

Figure 38. Top-down view of the PMD for a constant altitude experimental flight

A graph of the response for the distance traveled versus time is presented in Figure 39 for each axis. The red line represents the PMD flight, and it is observed that at the trailing edge of each WP section there is a change in direction to reorient to the next WP. The three plots for X, Y, and Z do not exceed 0.2 meters for any WP.

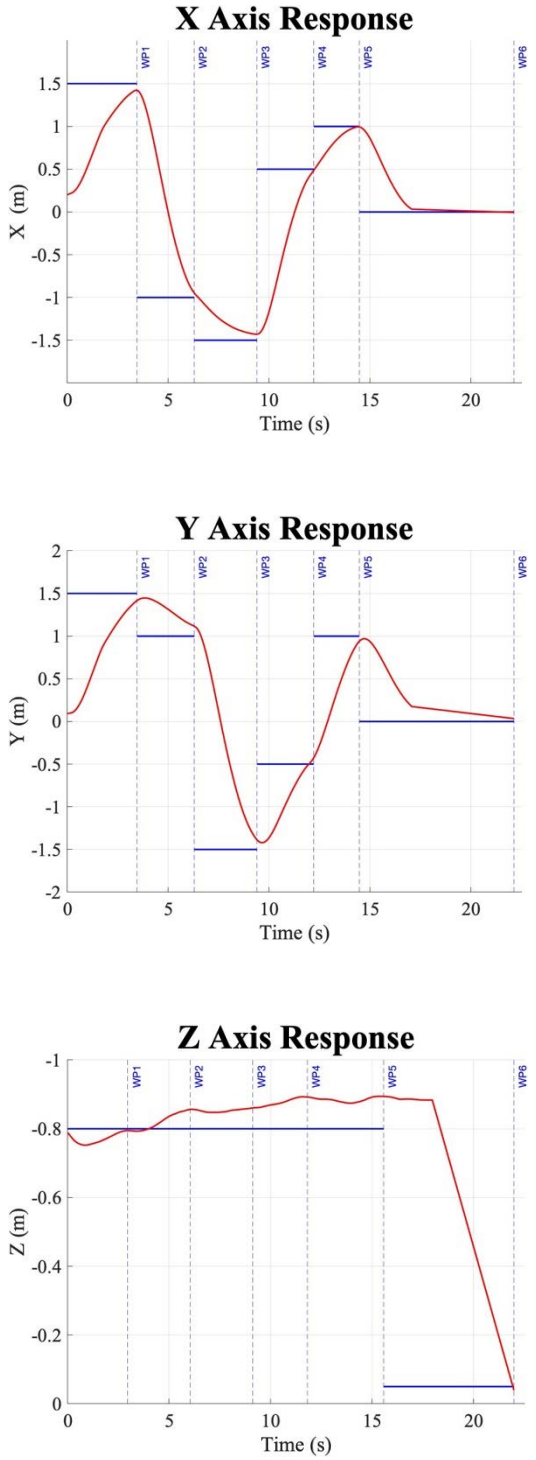

Figure 39. Three axes (X Y Z) distance response for the PMD with a constant altitude

To reach the desired WPs, the command inputs for the yaw, pitch, and roll are depicted in Figure 40. The commands are evaluated in degrees versus the total time in seconds taken for the experimental flight. Each WP is marked with a blue dashed line at the moment the system has determined it has met the desired WP tolerances. The control law limited the response movement to a maximum of 0.24 degrees.

Each plot has a distinct spike at the moment each WP is achieved. As in the simulated trial, this spike is attributed to the need for the system to adjust the PMD position to orient to the next WP. The values that are required to make the adjustments are noticeably larger than that of the simulated trial. This can be related to the supervisory control that is applied to the PMD control system. The PD controller outside of the internal PID controller allows the PMD to make more efficient adjustments. Also, the conditions for the experimental trial have other factors affecting the PMD flight performance that require the command inputs to compensate for those factors. Considerations, such as the battery voltage, interrupted air flow, and motor speed are a few such factors that contribute to the performance of the PMD.

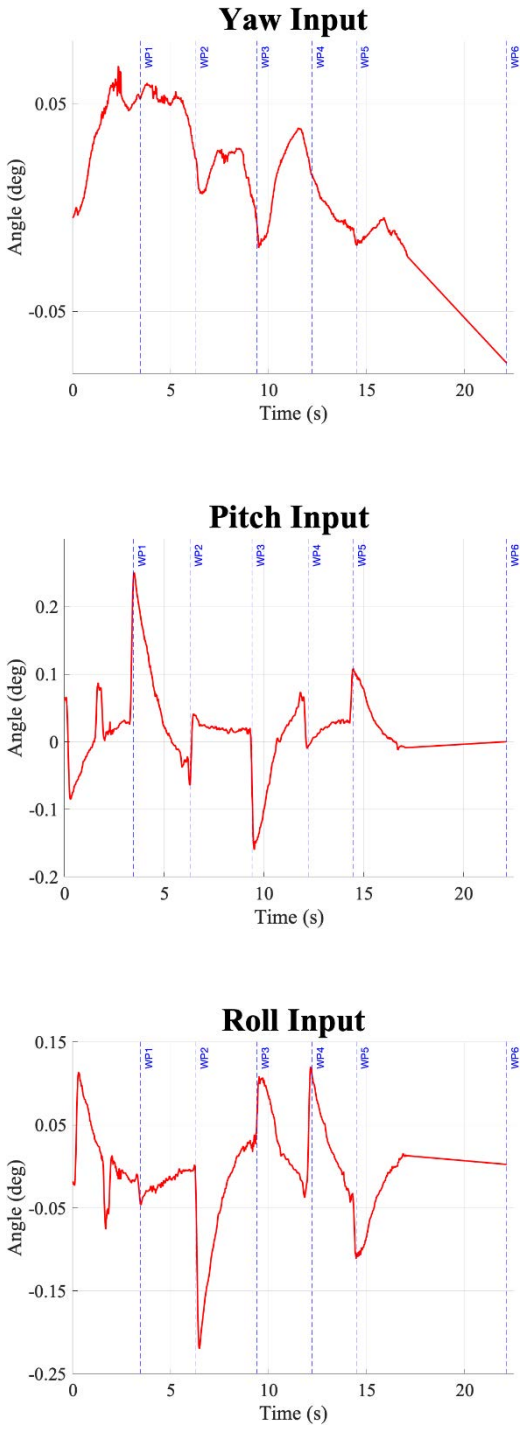

Figure 40. Three rotations (Y P R) for the PMD command inputs

Table 6 is compiled with the final percentage error value at the end of each WP flight iteration. The PMD is restricted to the values in Table 4 before it can proceed to the next WP goal. Each control parameter for the experimental flight of the PMD stayed within the required parameters to meet the goals of the research.

Table 6. Percent error for the PMD experimental constant altitude flight

Position	WP 1	WP 2	WP 3	WP 4	WP 5	WP 6
X	7.3389	12.6382	4.9410	4.4712	2.0200	0.0001
Y	9.6399	13.8731	11.3209	9.4157	17.5685	0.0003
Z	6.3454	2.5956	0.4941	2.7063	1.1229	6.2445
Yaw	6.1656	2.1533	4.6539	8.7738	8.4011	75.8566
Average Percentage Error	7.3724	7.8151	5.3525	6.3417	7.2781	0.0009

2. Varying Altitude Flight for the PMD

The final flight information that is presented for this thesis is the full flight to the desired WPs with varying degrees for all 6-DoF. The same plots that are used in the simulation and constant altitude are applied for the final trial. Figure 41 is the three-dimensional view for the PMD. It is observed that there is a distinct transition from the position movement to the altitude adjustment.

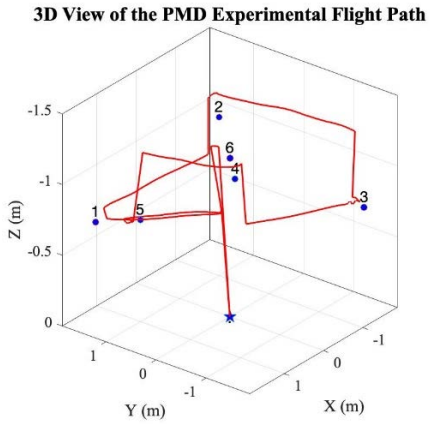

Figure 41. Flight path of the PMD during experimental flight

The top view of the flight path, Figure 42, resembles that of Figure 38 for constant altitude in that there is a curve at each WP transition. For this experimental trial with varying altitude, it can be observed that there is an abnormal flight path around WPs 2, 3, and 5. These irregularities are credited to the altitude adjustment that is now required. When the PMD transitions from the position mode to the vertical mode, the PMD has a slight drift when moving in the Z axis. The control laws that are applied adapt to the error and adjust the PMD position appropriately to reach each desired WP in an efficient manner.

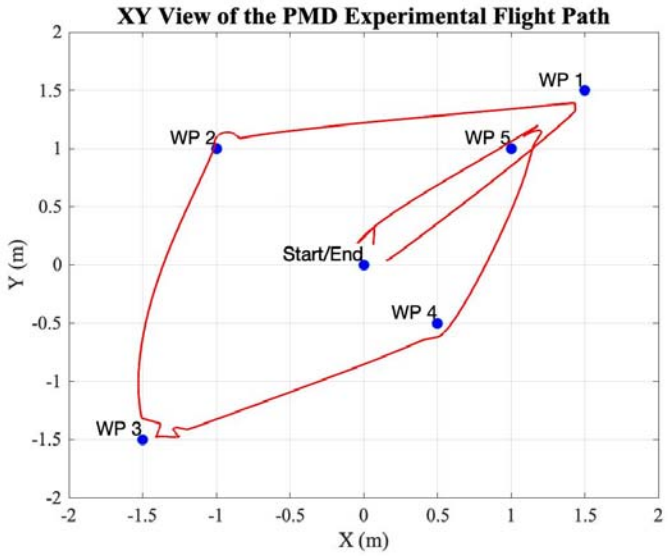

Figure 42. Top-down view of the PMD experimental flight path

Figure 43 is created to view the individual axis response. The PMD flight path, depicted in red, can be observed altering the direction at the trailing edge of each WP completion, annotated with a solid blue line.

Figure 44 of the yaw, pitch, and roll inputs presents the degrees of input versus time for the PMD flight. These plots maintain the same trend as the previous trials and have a spiked input at the end of each WP reached.

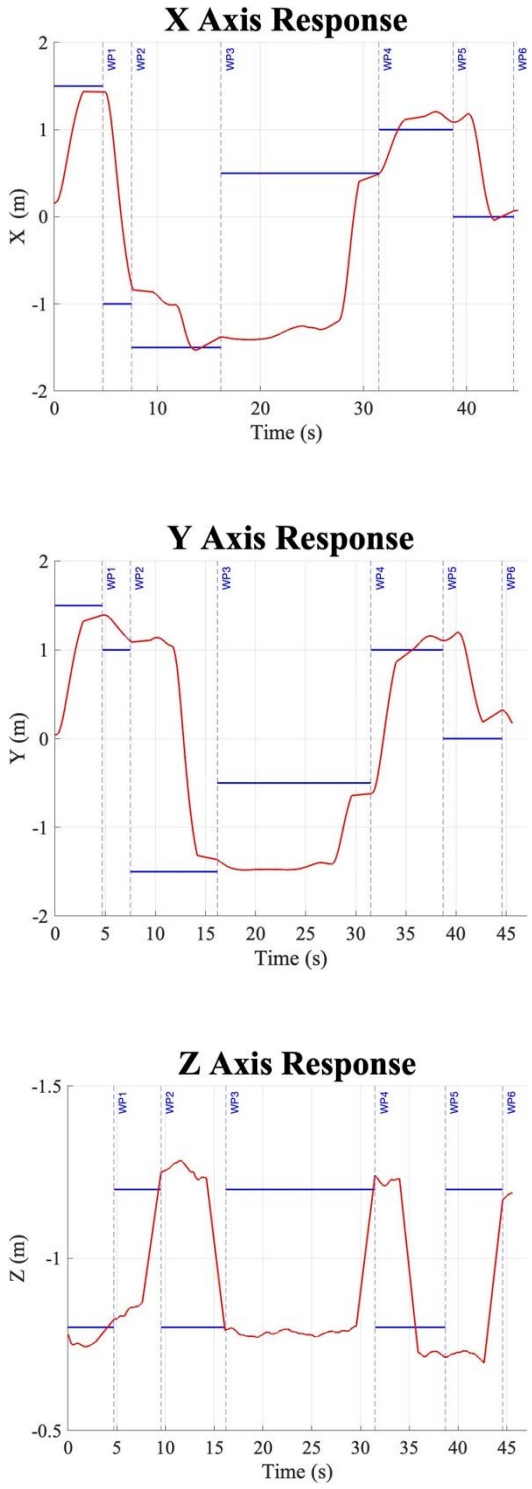

Figure 43. Three axes (X Y Z) distance response for the PMD

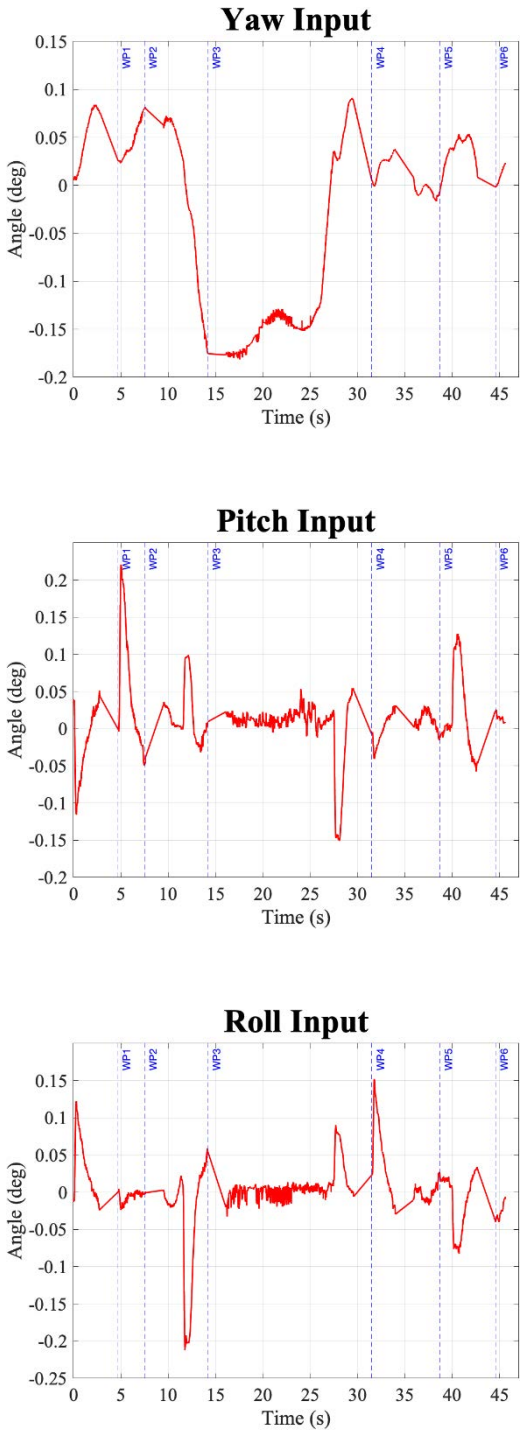

Figure 44. Three rotations (Y P R) for the PMD inputs

Table 7 is compiled with the final percentage error value at the end of each WP flight iteration. The PMD is restricted to the values in Table 4 before it can proceed to the next WP goal. Each control parameter for the experimental full flight of the PMD stayed within the required parameters to meet the goals of the research.

Table 7. Percent error for the PMD experimental full flight

Position	WP 1	WP 2	WP 3	WP 4	WP 5	WP 6
X	0.3417	10.0272	13.0426	0.3917	9.1327	0.0006
Y	0.1547	16.6599	1.5658	11.2828	11.0823	0.0001
Z	6.8073	0.0083	17.6462	5.6059	1.7259	1.9771
Yaw	3.6593	1.7052	0.6448	0.3110	8.3170	1.0128
Average Percentage Error	2.7407	7.1001	8.2248	4.3978	7.5645	0.0005

3. Trial Comparisons for the PMD

To compare the two experimental flights to the simulated flight, plots of the axis, overall flight path and time step are presented in Figure 45 through Figure 48.

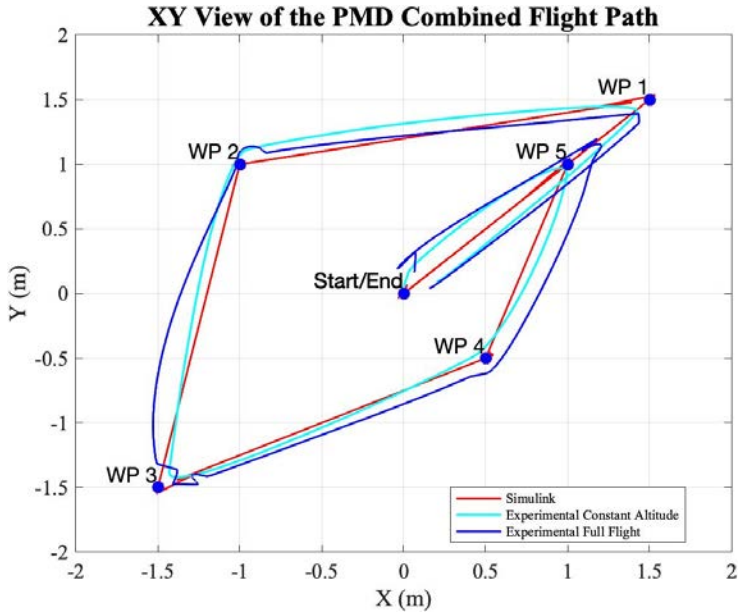

Figure 45. Top-down view of all three trials for the PMD

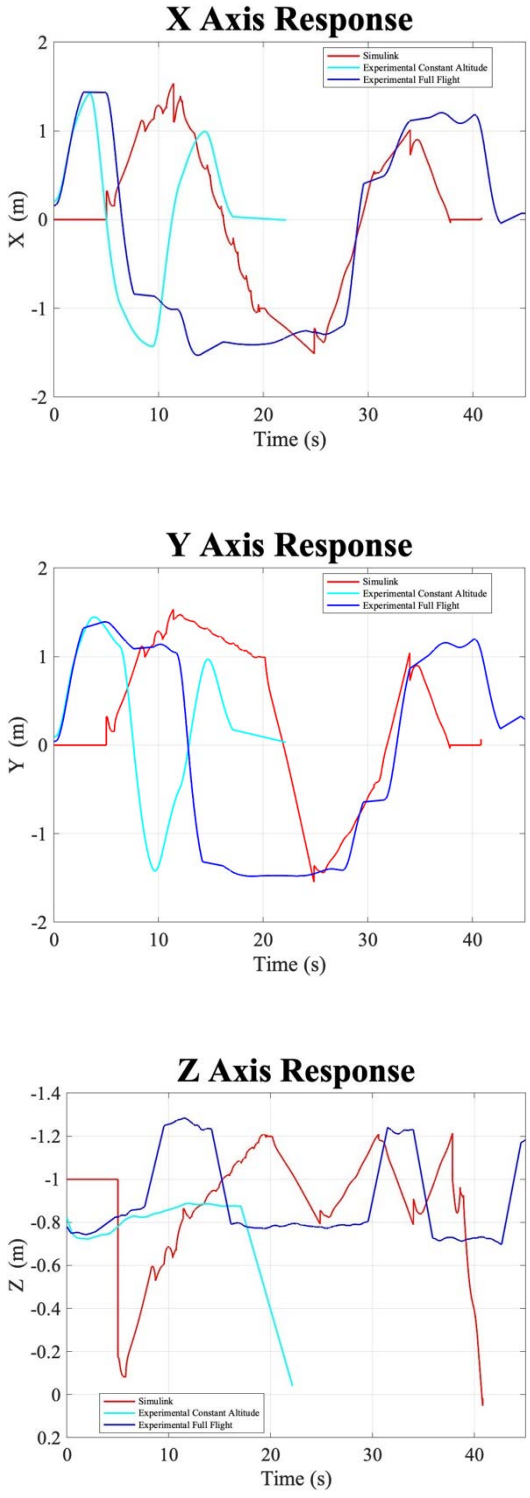

Figure 46. Combined plot of the three flight path trials for the PMD

The plots from the combined flight paths depict the similarities between the three trial flights. Each trial maintained the same basic flight path when viewed from a top-down view in Figure 45. There is very little error between the three paths. The main difference is noticed when Figure 46 is analyzed. The time for the PMD to complete a flight path with a constant altitude to all six of the desired WPs is half that of the simulated and full experimental flight. These values are viewed in cyan on all of the combined plots. Another attribute taken from this figure is that the experimental flight appears to have less noise and disturbance when completing the flight path.

To compare the performance of the simulation and the experiment further, two WPs are chosen to analyze the time constant (i.e., time required to reach 63% of the final value) for that section of the flight. Depicted in Figure 47 is the initial launch of the PMD from start to WP 1. The right plot analyzes a zoomed in section of the full plot on the left from 1.5 to 3.2 seconds. The three trials are analyzed from the X axis response, and the calculated time constant is very similar for all three trials. Table 8 has the calculated time constants for each trial flight for what the time step value is and the distance the PMD traveled to achieve the 63% for the PMD in order to go from zero to the final value of the WP. For this WP, the experimental full flight performed the best with the supervisory control implemented.

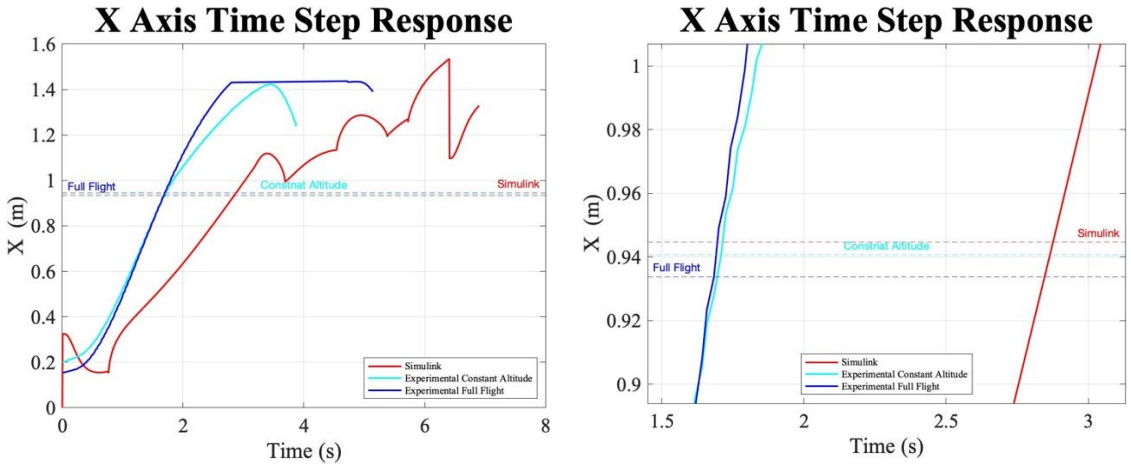

Figure 47. Time step response for the X axis from start to WP 1 for all three trials with the right as a zoomed in version from 1.5 to 3.2 seconds

Table 8. Time constant and the distance the PMD arrived at 63% at WP 1 for the three trial flights

	Simulink Full Flight	**Experimental Constant Altitude**	**Experimental Full Flight**
Time Constant	2.8753 (s)	1.7083 (s)	1.6083 (s)
Distance	0.9428 (m)	0.9406 (m)	0.9338 (m)

The second WP analyzed to compare the effectiveness of the supervisory control method using a PD control is WP 4. Figure 48 is viewed from WP 3 to WP 4 of time versus the distance traveled on the Y axis. A different axis response is viewed to ensure diversity in the data collected so as to avoid any favoritism with plots. From the data presented in Table 9, it is concluded that the experimental flight with constant altitude performed the best. There is a difference of about one second in the time constant between the experimental flights compared to the simulated flight. This difference can be attributed to some unknown gains within the PMD inner control loop. The SSPPM cannot be fully examined for the preprogrammed gains within the programming of the drone because a Bluetooth connection was not used in this research. The gains that are manipulated have been adjusted so that the experimental and simulation are as close as possible. The performance of both experimental flights with supervisory controllers applied performed ideally when compared to the simulated flight for the PMD. With the accuracy of the simulation to the experimental flights, the SSPPM can be used as a tool to further explore the PMD without relying on an experimental design space.

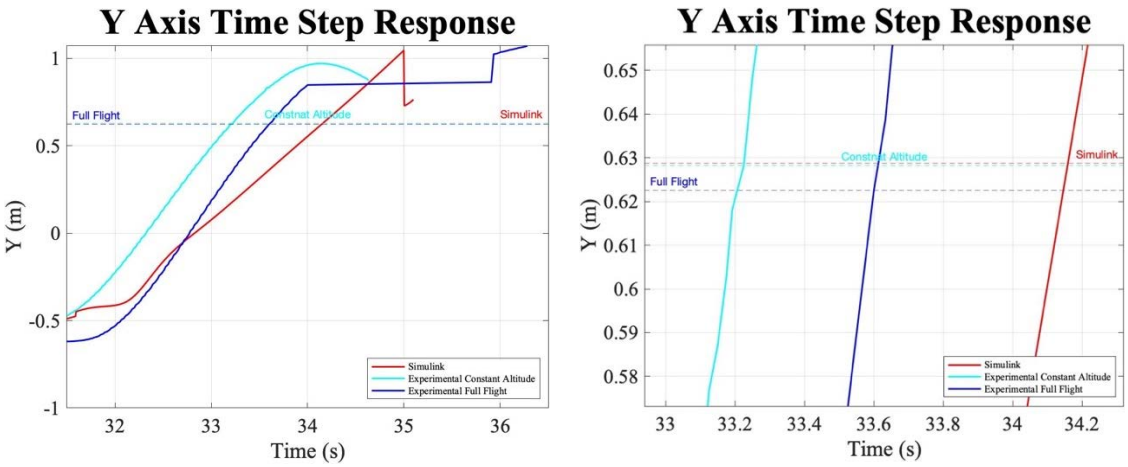

Figure 48. Time constant response for the Y axis from WP 3 to WP 4 for all three trials with the right as a zoomed in version from 33 to 34.3 seconds

Table 9. Time constant and the distance the PMD arrived at 63% at WP 4 for the three trial flights

	Simulink Full Flight	**Constant Altitude**	**Experimental Full Flight**
Time Constant	2.6625 (s)	1.7384 (s)	2.1011 (s)
Distance	0.6287 (m)	0.6225 (m)	0.6282 (m)

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION AND FUTURE WORK

As the demands from the Department of Defense and the civilian sector for UAVs increase, the necessity for efficient and reliable performance also broadens. Unknown terrain and denied environments make navigation tricky for military members. This thesis investigated whether a micro UAV could be adapted to have a better stability and control for use in path planning or target intercept as previously detailed in [5], [26], and [11]. A common issue with the use of micro UAVs is with the dependence on a ground base controller to guide them. To address the dependence issue, this thesis research explored the use of a supervisory PID controller. The controller was explored to provide the UAV an increased degree of autonomy. The MATLAB code that was developed for this thesis research was designed to allow the UAV to complete the desired mission without the reliance on a ground base controller. Utilizing the preprogrammed PID, MATLAB code with PD switch cases and the OptiTrack visual tracking system, all UAV motion was observed and the error from each WP was calculated accurately.

A. ASSESSMENT OF GOALS

The research objective was to determine if a supervisor controller can be used to provide autonomous operation of a quadrotor drone. This research was conducted through simulation using MATLAB/Simulink, as well as experimentation with the actual PMD. Simulations in Simulink are carried out to find a supervisor controller design that gives satisfactory performance in terms of speed and accuracy. Experiments with the actual PMD are also conducted to further validate the controller design. Comparisons between experiment and simulation results are also included.

A set of desired WPs were defined to set a benchmark for the PMD flight path. Each of the three flight trials executed a flight pattern while the time and percent error of the simulated and experimental flight paths were recorded. A summary of the flight paths from the flight trials was compiled, and the paths were compared. The flight statistics, contained in the summary, highlight the strengths and weaknesses of each flight trial.

The trial with the best response time proved to be the experimental trial flight with the supervisory control implemented. The designed supervisory controller, using PD control law, allowed the PMD to reach each WP on average 0.5 seconds quicker than without.

It was observed that the experimental flight with a supervisory control with constant altitude resulted in the best overall values. The PMD completed the flight pattern in half the time that it took to do any altitude adjustments. The controller also had an average time step of 1.4 seconds for the six WPs defined.

The trial with the lowest average error when reaching the desired WPs was the simulated trial flight. This data demonstrates that in perfect conditions, the PMD control law is sufficient to reach desired WPs

B. LIMITATIONS

This research encountered multiple limitations. These limitations included connectivity issues, limited space for trials, lack of hardware support, and poor battery output reliability.

The limitation that had the greatest impact on this research was the connectivity issue. This research required four systems to maintain connection with each other at all times. The Wi-Fi dongle would occasionally loose connectivity for a moment with the PMD. When this happened in flight, the PMD would not receive the updated information for the position. Without this information, the PMD would maintain its current direction and speed of flight, resulting in multiple crashes into walls and safety nets.

In addition to the connectivity issue, the size of the flight area had a small impact on our analysis. With the small physical area, there was little room for any error while conducting trial flights. The flight area was limited to a five-meter distance in the X and Y and a two-meter distance for the Z. These constraints limited the amount of testing that could be performed to definitively determine the effectiveness of the supervisory control that was implemented.

An initial research goal was to control the PMD with both Bluetooth and Wi-Fi. This was not possible due to the compatibility issues between operating systems and the PMD requirements. The required Remote Network Driver Interface Specification (RNDIS) was not supported on the systems that were used for this research. Without the driver, the SSPPM was unable to build and deploy the algorithms from the Simulink model to test on the PMD.

The final limitation that was encountered involved the PMD battery packs. The PMD was rated to be able to fly for 10 minutes with a full battery. This proved to be mostly accurate, but the performance of the PMD was significantly degraded as the battery level decreased. It was annotated during the research that when a battery fell below 25% for a trial flight, the PMD would struggle to obtain the desired thrust to move from one WP to the next. Conversely, it was also observed that on a full battery, the PMD would over perform and provided more thrust than necessary occasionally overshooting WPs and exceeding limitations of the PMD, which are detailed in Table 4. To avoid overshooting and exceeding limitations, a non-inverting buck-boost converter can be applied to allow the PMD to increase or decrease voltage. The converter will allow for active voltage stabilization to tighten the voltage tolerance. The addition of the converter has the potential to decrease the PMD efficiency and add weight to the platform but will make the PMD easier to operate.

C. RECOMMENDATIONS FOR FUTURE WORK

The following recommendations for future work are based on observations made during the testing of the PMD in each trial summarized in the previous chapters. The recommendations also coincide with aspects of the research that are not a part of the overall scope of testing for this research.

One opportunity for future work would be to work on incorporating an integrator block to the controller model. This would allow for a more accurate vertical position error estimation. The incorporation allows for the system to establish the WP destination in a shorter amount of time and with more efficiency.

With a capable control system designed, the estimation of energy consumption by the PMD could be enhanced by using motor-speed control during flight planning. For this research, the motors were not adjusted and were allowed to use full thrust when moving. Using an energy efficient method to dictate when the PMD can use a specified amount of energy could significantly enhance the accuracy that the PMD has when reaching desired WPs. This will also extend the battery life, allowing the PMD to fly for a longer time duration.

Another avenue of research would be to implement the Bluetooth Simulink model. This will require more research into the driver support packages for the OptiTrack and computer systems that are available for the research.

A more reliable Wi-Fi dongle could solve the connectivity issue. With a reliable connection, flight tests could be performed with more confidence, allowing for better adjustments to the control gains that are applied. Instead of using Wi-Fi, a radio-controlled communication can be implemented. This method of communication is commonly used with remote controlled modeled airplanes allowing for a substantially greater communication range than Wi-Fi. The potential drawback to using radio-controlled communications would be reduced information flow, in the for of control, to the PMD. The radio controlled communication would gain communications range, but lose information bandwidth compared to Wi-Fi.

Another opportunity for future research to expand on is the use of the PMD to pursue pursuit guidance with an additional PMD. The control law would be implemented onto a PMD to conduct flight patterns. While this drone conducts a flight pattern a secondary drone would identify and then intercept the first PMD. This research is currently prevalent in many different industries, as well as in the military. Such issues are identified in [10], [34],[35], and [36].

APPENDIX A. DRONE NAVIGATION

```
clear;   clc;                close all

%% set up global variable to control the while-loop
global droneControlLoopOperator

%% create the unicontrol to manage the control loop from the "ABORT" button

   figHandle = figure;
   figHandle.Position = [1770  930  117  66];
   figHandle.ToolBar = 'none';
   buttonHandle =  uicontrol(figHandle,
'Style','pushButton','Callback',@pushbutton1_Callback);
   buttonHandle.String = 'ABORT';
   drawnow;

%% Establish Communication with MOTIVE directly.
optitrack_connector

%% Establish communication with parrot Mambo
fprintf('\n\nConnecting to Mambo...\n')
p = parrot();
fprintf('Connected to %s\n', p.ID)
fprintf('Battery Level is %d%%\n', p.BatteryLevel)

%% Define desired tolerances and gains
yawTol = 3.0 * pi/180;      % yaw threshold is 3 degrees
positionTol = 0.2;          % position error 0.2 m
turnDur = 0.3;              % duration of input 300 ms
vertDur = 0.8;              % duration of 0.8 sec for each vertical climb

% Positional Gain Values
kYaw = 0.9;                 % steering gain to turn drone CW/CCW,
kPitch = -0.086;            % gain is negative bc this is required to
                            get pos. displacement
kRoll = 0.086;              % Optimized at 0.086
kVert = 1;

% Derivative Gain Values
kD_Pitch = -0.1;            % gain is negative bc this is required to
                            get pos. displacement
kD_Roll = 0.1;
kD_Yaw = .3*0.6;
```

```
kD_Vert = 0.1;

%% Goal waypoint
                % goal location and orientation [x,y,z, roll, pitch, yaw]
goalDesired = [ 1.5    -1.0    -1.5    0.5    1.0    0.0;
                1.5     1.0    -1.5   -0.5    1.0    0.0;
               -0.8    -1.2    -0.8   -1.2   -0.8   -1.2;
                0       0       0      0      0      0;
                0       0       0      0      0      0;
  deg2rad(0) deg2rad(0) deg2rad(0) deg2rad(0) deg2rad(0)  deg2rad(0)];


gD_s = size(goalDesired);
waypoint_itr = {};
time = {};
t = 0;

for itr = 1:gD_s(2)
    xyDesired(1:2,itr) =    goalDesired(1:2, itr);
    vertDesired(3,itr) =    goalDesired(3, itr);
    rollDesired(4,itr) =    goalDesired(4,itr);
    pitchDesired(5,itr) =   goalDesired(5,itr);
    yawDesired(6,itr) =     goalDesired(6, itr);

    stateArray = zeros(7,0);    %[X;Y;Z; roll;pitch;yaw; Timestamp
    errorArray = zeros(4,0);    % store roll, pitch, yaw command inputs
    commandArray = zeros(4,0);  % to store pitch and roll commands
    TotalError = zeros(1,0);
    old_timestamp =  0;         % previous time for delta T calculation
                        for Derivative
    old_xy_Error =   0;         % previous position error for
                        Deravitive calc
    old_yaw_Error =  0;
    old_vert_Error = 0;

%% start the drone control loop
droneControlLoopOperator = true;
drone_mode = 'TAKE_OFF_MODE';   % initialize the switch-case to

TAKE_OFF_MODE
inc = 0;
while droneControlLoopOperator
    t = t+1;
    inc = inc+1;
    if inc == 1
```

```
    takeoff(p);
end

% Read current position
% state is [X;Y;Z; roll; pitch; yaw; TIMESTAMP]
state = optitrack_getDronePose;
stateArray(:,inc) = state;
positionActual = state(1:2);
yawActual = state(6);
vertActual = state(3);
timestamp = state(7);

% to rotate X,Y from world frame to robot frame
Tw2r = [cos(yawActual), sin(yawActual); -sin(yawActual), cos(yawActual)];

% to handle array storage at the end of the loop
rollCmd = NaN; pitchCmd = NaN;  yawCmd = NaN; vertCmd =NaN;

% Time Derivative for kD gains
deltaT = timestamp - old_timestamp;

    if deltaT < eps
        deltaT = 0.008301;        % Average calculated time step
    end

old_timestamp = timestamp;

% Compute the errors
     % Yaw Error
    yawError = wrapToPi(yawDesired(1,itr) - yawActual);
    yawD_Error = (yawError-old_yaw_Error)/deltaT;
    old_yaw_Error = yawError;

     % compute the yaw commands
    yawCmd = kYaw*yawError+kD_Yaw*yawD_Error; % + made it crash

    if abs(yawCmd) > pi
        yawCmd = sign(yawCmd)*pi;
    end

      % Position Error
    xyError = xyDesired(1:2,itr) - positionActual;
    xyD_Error = (xyError - old_xy_Error)/deltaT;
    old_xy_Error = xyError;
```

```
 % compute the pitch commands
roll_pitch_cmd = (Tw2r)*xyError;  %error in robot frame

pitchCmd = kPitch*roll_pitch_cmd(1) + kD_Pitch*xyD_Error(1);

if abs(pitchCmd) > 0.43633              % limitations of Parrot Drone
   pitchCmd = sign(pitchCmd)*0.43633;
end

% compute the roll commands
rollCmd = kRoll*roll_pitch_cmd(2)+ kD_Roll*xyD_Error(2);

if abs(rollCmd) > 0.43633               % limitations of Parrot Drone
   rollCmd = sign(rollCmd)*0.43633;
end

 % Altitude Error
vertError = vertDesired(3,itr) - vertActual;
vertD_Error = (vertError - old_vert_Error)/deltaT;
old_vert_Error = vertError;

vertCmd = kVert*vertError+kD_Vert*vertD_Error;

 if vertCmd < -2 ||  vertCmd > 2      % Parrot Limit of +-2 m/s
   vertCmd = 1;
 end

totalError = [xyError; yawError; vertError];

% store data for post analysis
  errorArray(:,inc) = [xyError; yawError; vertError];
  commandArray(:,inc) = [pitchCmd; rollCmd; yawCmd; vertCmd];
  TotalError(:,inc) = norm(totalError);

% format the output to display on the screen
%      fprintf('sample %d:  X,Y,Z   %5.2f  %5.2f   %5.2f\n', inc, positionActual(1),
positionActual(2), vertActual);

  switch drone_mode

    case 'TAKE_OFF_MODE'
      if vertActual > -0.5
         pause(0.1);    % wait 100 ms to give drone a chance to rise
         disp('waiting for altitude')
      else
```

```
            drone_mode = 'YAW_MODE'; % if we are above 1/2 m AGL, switch to
NAV_MODE
            disp('Take-off mode complete, switching to Yaw mode')
          end

      case 'YAW_MODE'
          if norm(yawError) > yawTol
              move(p, turnDur, 'RotationSpeed', yawCmd);
             % pause(turnDur);
          else
              drone_mode = 'POS_MODE';
              disp('Yaw mode complete, switching to Position Mode')
              pos_counter = 0;
          end

      case 'POS_MODE'
%           tic
          move(p, turnDur, 'pitch',pitchCmd, 'roll',rollCmd);
%           toc
          pause(0.055*turnDur);         % Needed or the drone will fly into the wall
          if norm(xyError) > 0.4
              drone_mode = 'POS_MODE';
%               disp(sprintf('XY Error of %g m tolerance at %g m, switching to Pos
mode.\n',norm(xyError), positionTol))
              pos_counter = 0;
          elseif norm(xyError) < 0.4 && norm(xyError) > 0.2
              move(p, turnDur/2, 'pitch',pitchCmd*2, 'roll',rollCmd*2);
              disp(sprintf('XY Error of %g m tolerance at %g m\n',norm(xyError),
positionTol))

          else
              move(p, 1.5*turnDur, 'pitch',0, 'roll',0, 'RotationSpeed', 0);
              pos_counter = pos_counter + 1;
              if pos_counter == 1
                  drone_mode = 'VERT_MODE';
%                   drone_mode = 'HOVER';
                  hover_counter = 0;
                  vert_counter = 0;
                  disp('Pos mode complete, switching to Vert mode')
              end
          end

      case 'VERT_MODE'
          move(p, vertDur,'VerticalSpeed', vertCmd);
          if norm(vertError) > 0.4
```

```
            drone_mode = 'POS_MODE';
            vert_counter = 0;
            disp(sprintf('Vertical Error outside required limits of %g m at %g m, switching
to Pos mode.\n',positionTol, vertError))
        else
            move(p, 1.5*turnDur, 'pitch',0, 'roll',0, 'RotationSpeed', 0,'VerticalSpeed',0);
             vert_counter = vert_counter + 1;
             if vert_counter == 1
                drone_mode = 'HOVER';
                hover_counter = 0;
                disp('Vert mode complete, switching to Hover')
             end
        end

    case 'HOVER'
        % Fly steady above waypoint target
         if norm(totalError) > 0.2
            drone_mode = 'YAW_MODE';
            disp(sprintf('Total Error of %g m  off at %g m, switching to Yaw mode.\n',
positionTol,norm(totalError)))
            hover_counter = 0;
            pos_counter = 0;
        else
            hover_counter = hover_counter + 1
            move(p, 1.5*turnDur, 'pitch',0, 'roll',0, 'RotationSpeed', 0);
            if itr < gD_s(2)
               drone_mode = 'YAW_MODE';
               disp(sprintf('Hover mode pending, switching to New Iteration %d of
%d',1+itr,gD_s(2)))
                  droneControlLoopOperator = false; % break out of this loop first, then
land the drone.
            else
               drone_mode = 'LAND_MODE';
               disp('Hover mode complete, switching to Land mode')
               land_counter = 0;
               land(p);
            end
        end

    case 'LAND_MODE'

         land_counter = land_counter +1;
         land(p);
         if land_counter == 5
```

```
                droneControlLoopOperator = false; % break out of this loop first, then land
the drone.
            end

    end    % switch-case

end     % while-loop
waypoint_itr{itr} = stateArray;
time{itr} = t;
end     % for loop of iterations for way points

fprintf('Battery End Level is %d%%\n', p.BatteryLevel)

land(p);  % in case we had to abort while-loop above
clear p
close(figHandle)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. OPTITRACK CONNECTOR

```
function []=optitrack_connector()
% Run this function at the start of your user script to connect to the
% Optitrack system.  Be sure MATLAB path has
%     C:/Users/localadmin/Desktop/myDrones
%     C:\Users\localadmin\Desktop\Natnetsdk30\NatNetSDK\Samples\MATLAB

global natnetclient

fprintf( 'Optitrack Connector Starting...\n' )

   % create an instance of the natnet client class
   fprintf( '\tCreating natnet class object\n' )
   natnetclient = natnet;

   % connect the client to the server (multicast over local loopback) -
   % modify for your network
   fprintf( '\tConnecting to the server\n' )
   natnetclient.HostIP = '127.0.0.1';
   natnetclient.ClientIP = '127.0.0.1';
   natnetclient.ConnectionType = 'Multicast';
   natnetclient.connect;
      if ( natnetclient.IsConnected == 0 )
         fprintf( 'Client failed to connect\n' )
         fprintf( '\tMake sure the host is connected to the network\n' )
         fprintf( '\tand that the host and client IP addresses are correct\n\n' )
         return
      else
         fprintf( '\tNatNet client is connected to the Optitrack system.\n')
      end
   % get the asset descriptions for the asset names
   model = natnetclient.getModelDescription;
      if ( model.RigidBodyCount < 1 )
         fprintf( 'No Rigid Bodies found.\n' )
         return
      else
         mm = model.RigidBody.Name;
         nn = model.MarkerSet.MarkerCount;
         fprintf('\tTracking %s with %2d markers.\n\n',mm,nn );

      end
end  % end function
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. OPTITRACK DRONE ORIENTATION ADJUSTMENT

```
function [poseArray] = optitrack_getDronePose()
% Put this functio in your user script to get the pose of the drone. The
% output POSEARRAY has the following format:X,Y,Z coordinates in meters
%       roll, pitch, yaw in radians
% The coordiante system for the drone cage is +x is East, +y is South,
% and +z is Down. That following code converts the Optitrack frame data
% into our preferred orientation to match the Mambo drone.
%
% Be sure to have MATLAB path set as described in the accompanying function
% OPTITRACK_CONNECTOR.  See help comments.
global natnetclient

rotateQuatX = [cos(pi/4)  sin(pi/4) 0 0]; % use to rotate OptiTrack quaternion
data = natnetclient.getFrame; % method to get current frame

      if (isempty(data.RigidBody(1)))
         fprintf( '\tPacket is empty/stale\n' )
         fprintf( '\tMake sure the server is in Live mode or playing in playback\n\n')
         return
      end

      % get the Timestamp
      time = data.Timestamp;
      % work out Euler angles
      u(1) = double( data.RigidBody( 1 ).qw );   % 1 is the number of rigid bodies
      u(2) = double( data.RigidBody( 1 ).qx );
      u(3) = double( data.RigidBody( 1 ).qy );
      u(4) = double( data.RigidBody( 1 ).qz );
      u = u/norm(u);
       newU = rotate_v_by_q(u, rotateQuatX).';
      angles = quat2eul(newU);          % outputs orientation as [yaw, pitch, roll]

      X = double( data.RigidBody( 1 ).x );
      Y = double( data.RigidBody( 1 ).z );
      Z = double( -data.RigidBody( 1 ).y );
      phi = angles(3);
      theta = -angles(2);          %**** temp fix with a negative sign*****
      psi = -angles(1);
      poseArray = [X; Y; Z; phi; theta; psi; time];
end   % function end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. QUATERION MULTIPLICATION FUNCTION

```
function qout=q_mult2(p,q)
% code provided by James Calusdian from []

P_mat = [p(1) -p(2) -p(3) -p(4);
        p(2)  p(1) -p(4)  p(3);
        p(3)  p(4)  p(1) -p(2);
        p(4) -p(3)  p(2)  p(1)];
qout = P_mat*q;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. MOTIVE CONNECTION TO MATLAB FOR DRONE POSITION DATA

```
%% Script to provide pose data from Motive
% modified from Dr. James Calusdian's code

fprintf( 'Creating natnet class object\n' )
natnetclient = natnet;

% connect the client to the server (multicast over local loopback) -
% modify for your network
fprintf( 'Connecting to the server\n' )
natnetclient.HostIP = '127.0.0.1';
natnetclient.ClientIP = '127.0.0.1';
natnetclient.ConnectionType = 'Multicast';
natnetclient.connect;
if ( natnetclient.IsConnected == 0 )
    fprintf( 'Client failed to connect\n' )
    fprintf( '\tMake sure the host is connected to the network\n' )
    fprintf( '\tand that the host and client IP addresses are correct\n\n' )
    return
end

% get the asset descriptions for the asset names
model = natnetclient.getModelDescription;
if ( model.RigidBodyCount < 1 )
    return
end

u = [0 0 0 0];   % define a quaternion

% tcpipServer = tcpip('0.0.0.0',80,'NetworkRole','Server');
% set(tcpipServer,'OutputBufferSize',48);
% fopen(tcpipServer);
% state_array = zeros(6, 30);  % used for static tests
% t = zeros(1,30);
% inc = 1;
rotateQuatX = [cos(pi/4)  sin(pi/4) 0 0]; % use to rotate OptiTrack quaternion
while  1%inc < 30
    %java.lang.Thread.sleep( 996 );
%     tic
    data = natnetclient.getFrame; % method to get current frame

    if (isempty(data.RigidBody(1)))
```

```
        fprintf( '\tPacket is empty/stale\n' )
        fprintf( '\tMake sure the server is in Live mode or playing in playback\n\n')
        return
    end


    % work out Euler angles
    u(1) = data.RigidBody( 1 ).qw;   % 1 is the number of rigid bodies
    u(2) = data.RigidBody( 1 ).qx;
    u(3) = data.RigidBody( 1 ).qy;
    u(4) = data.RigidBody( 1 ).qz;
    u = u/norm(u);
    if u(1) < 0
        u = -u;
    end

    newU = rotate_v_by_q(u, rotateQuatX).';
    u
    newU
    angles = quat2eul(newU);          % outputs orientation as [yaw, pitch, roll]
    %angles = quat2eul(u);
    X = data.RigidBody( 1 ).x;
    Y = data.RigidBody( 1 ).z;
    Z = -data.RigidBody( 1 ).y;


    phi = angles(3) * 180/pi;
    theta = -angles(2) * 180/pi;          %**** temp fix with a negative sign*****
    psi = -angles(1) * 180/pi;
    state = [X, Y, Z, phi, theta, psi].'
    %fwrite(tcpipServer,state(:),'double');
%       state_array(:,inc)=state;
%       inc = inc+1;
%
%       t(inc) = toc;
  pause(0.1);
 end
% fclose(tcpipServer);
```

APPENDIX F. ROTATION FUNCTION

```
function u=rotate_v_by_q(v,q)
% code provided by James Calusdian from []

q_inv= [q(1) -q(2) -q(3) -q(4)]';

u = q_mult2(q,q_mult2(v,q_inv));
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. MATLAB PLOT SCRIPT FOR EXPERIMENTAL TRIAL FLIGHT

```
clear;          clc;          close all

load("8_5_Waypoint_Trial5_VERTFULL.mat")

waypoint1 = horzcat(waypoint_itr{1});
waypoint2 = horzcat(waypoint_itr{2});
waypoint3 = horzcat(waypoint_itr{3});
waypoint4 = horzcat(waypoint_itr{4});
waypoint5 = horzcat(waypoint_itr{5});
waypoint6 = horzcat(waypoint_itr{6});


start = 0;
first = 6.0008;
second = 9.85;
third = 16.5;
fourth = 32.83;
fifth = 39.98;
six = 44.5;
stop = six;

waypoints = horzcat(waypoint_itr{:});

xaxis = waypoints(1,:);
yaxis = waypoints(2,:);
zaxis = waypoints(3,:);
roll = waypoints(4,:);
pitch = waypoints(5,:);
yaw = waypoints(6,:);
timeD = waypoints(7,:);
timeDe = diff(timeD);
time =cumsum(timeDe);

Desires Way Points for PMD
figure()
x = [0 1.5 -1 -1.5 0.5 1.0 0];
y = [0 1.5 1.0 -1.5 -0.5 1.0 0];
z = [0 -0.8 -1.2 -0.8 -1.2 -0.8 -1.2];
plot(x,y,'LineWidth',1.5)
ax =gca;
set(gca, 'FontName', 'Times New Roman')
```

```
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
hold on

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','r')
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
grid on
ylim([-2 2])
xlim([-2 2])
title('Desired Waypoints for the PMD','FontSize',18,"FontWeight","bold")
xlabel('X (m)')
ylabel('Y (m)')
```

3D View of PMD

```
figure()
labels1 = {'.','1','2','3','4','5','6'};
textscatter3(x,y,z,labels1,"FontSize",14)
hold on
plot3(0,0,0,'p','MarkerFaceColor','b','MarkerSize',12)
hold on
plot3(goalDesired(1,1),goalDesired(2,1),goalDesired(3,1)+.08,'o','MarkerFaceColor','b')
hold on
plot3(goalDesired(1,2),goalDesired(2,2),goalDesired(3,2)+.08,'o','MarkerFaceColor','b')
hold on
plot3(goalDesired(1,3),goalDesired(2,3),goalDesired(3,3)+.08,'o','MarkerFaceColor','b')
hold on
plot3(goalDesired(1,4),goalDesired(2,4),goalDesired(3,4)+.08,'o','MarkerFaceColor','b')
hold on
plot3(goalDesired(1,5),goalDesired(2,5),goalDesired(3,5)+.08,'o','MarkerFaceColor','b')
hold on
plot3(goalDesired(1,6),goalDesired(2,6),goalDesired(3,6)+.08,'o','MarkerFaceColor','b')
hold on

plot3(xaxis,yaxis,zaxis,'r','LineWidth',1.5)
hold on
plot3([0 0 0.06627],[0;0;0.1681],[0 0 -1.19],'r','LineWidth',1.5)
hold on
plot3([0 0 0.1544],[0;0;0.04202],[0 0 -0.7805],'r','LineWidth',1.5)
title('3D View of the PMD Experimental Flight Path','FontSize',18,"FontWeight","bold")
grid on
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')
zlim([-1.5 0])
```

```
yticks(-1 : 1 : 2);
view([-51.81 27.39])

set(gca, 'Zdir','reverse')
set(gca, 'Xdir','reverse')
set(gca, 'FontName', 'Times New Roman')
ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
ax.ZAxis.FontSize = 14;
```

XY Response for the PMD

```
figure()

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
hold on
plot(xaxis(1,:), yaxis(1,:),'r','LineWidth',1.5)
grid on
title('XY View of the PMD Experimental Flight Path','FontSize',18,"FontWeight","bold")
xlabel('X (m)')
ylabel('Y (m)')
grid on
ylim([-2 2])
xlim([-2 2])

ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
set(gca, 'FontName', 'Times New Roman')
```

X Response

```
figure()
set(gca, 'Zdir','normal')
xline(time(129),'--b',{'WP1'})
hold on
xline(time(255),'--b',{'WP2'})
xline(time(851),'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
line([0,          time(129)], [xyDesired(1,1),xyDesired(1,1)],'Color','b','LineWidth',1.5);
line([time(129),   time(255)], [xyDesired(1,2),xyDesired(1,2)],'Color','b','LineWidth',1.5);
line([time(255),   16.17], [xyDesired(1,3),xyDesired(1,3)],'Color','b','LineWidth',1.5);
```

```
line([16.17,  time(2571)], [xyDesired(1,4),xyDesired(1,4)],'Color','b','LineWidth',1.5);
line([time(2571), time(2792)],
[xyDesired(1,5),xyDesired(1,5)],'Color','b','LineWidth',1.5);
line([time(2792), time(2971)],
[xyDesired(1,6),xyDesired(1,6)],'Color','b','LineWidth',1.5);
plot(time,xaxis(2:end),'r','LineWidth',1.5)

grid on
title('X Axis Response','FontSize',32,''FontWeight'',''bold'')
xlabel('Time (s)')
ylabel('X  (m)')
ylim([-2 2])
xlim([0 45])
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-2 : 1 : 2);
```

Y Response

```
figure()

xline(time(129),'--b',{'WP1'})
hold on
xline(time(255),'--b',{'WP2'})
xline(16.17,'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
line([0,          time(129)], [xyDesired(2,1),xyDesired(2,1)],'Color','b','LineWidth',1.5);
line([time(129),   time(255)], [xyDesired(2,2),xyDesired(2,2)],'Color','b','LineWidth',1.5);
line([time(255),   16.17], [xyDesired(2,3),xyDesired(2,3)],'Color','b','LineWidth',1.5);
line([16.17,  time(2571)], [xyDesired(2,4),xyDesired(2,4)],'Color','b','LineWidth',1.5);
line([time(2571), time(2792)],
[xyDesired(2,5),xyDesired(2,5)],'Color','b','LineWidth',1.5);
line([time(2792), time(2971)],
[xyDesired(2,6),xyDesired(2,6)],'Color','b','LineWidth',1.5);
plot(time,yaxis(2:end),'r','LineWidth',1.5)

grid on
title('Y Axis Response','FontSize',32,''FontWeight'',''bold'')
xlabel('Time (s)')
ylabel('Y (m)')
```

```
ylim([-2 2])
xticks(0:5:50)
xlim([0 47])
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-2 : 1 : 2);
```

Z Response

```
figure()

set(gca, 'Ydir','reverse')
xline(time(129),'--b',{'WP1'})
hold on
xline(time(260),'--b',{'WP2'})
xline(16.17,'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
line([0,          time(129)], [vertDesired(3,1),vertDesired(3,1)],'Color','b','LineWidth',1.5)
line([time(129),   time(260)],
[vertDesired(3,2),vertDesired(3,2)],'Color','b','LineWidth',1.5)
line([time(260),   16.17], [vertDesired(3,3),vertDesired(3,3)],'Color','b','LineWidth',1.5)
line([16.17,  time(2571)], [vertDesired(3,4),vertDesired(3,4)],'Color','b','LineWidth',1.5)
line([time(2571), time(2792)],
[vertDesired(3,5),vertDesired(3,5)],'Color','b','LineWidth',1.5)
line([time(2792), time(2971)],
[vertDesired(3,6),vertDesired(3,6)],'Color','b','LineWidth',1.5)
plot(time,zaxis(2:end),'r','LineWidth',1.5)

grid on
title('Z Axis Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Z (m)')
ylim([-1.5 -0.5])
xlim([0 47])
xticks(0:5:50)

hold off

ax =gca;
ax.XAxis.FontSize = 18;
```

```
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
```

Roll Response

```
figure()

plot(time,roll(2:end),'r','LineWidth',1.5)
grid on
xline(time(129),'--b',{'WP1'})
hold on
xline(time(255),'--b',{'WP2'})
xline(time(847),'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
title('Roll Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
xlim([0 47])
xticks(0:5:50)

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
set(gca, 'YDir','normal')
```

Pitch Response

```
figure()

plot(time,pitch(2:end),'r','LineWidth',1.5)
grid on
xline(time(129),'--b',{'WP1'})
hold on
xline(time(255),'--b',{'WP2'})
xline(time(847),'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
title('Pitch Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
ylim([-0.2 0.25])
xlim([0 47])
xticks(0:5:50)
```

```
ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
```

Yaw Response

```
figure()

plot(time,yaw(2:end),'r','LineWidth',1.5)
grid on
xline(time(129),'--b',{'WP1'})
hold on
xline(time(255),'--b',{'WP2'})
xline(time(847),'--b',{'WP3'})
xline(time(2571),'--b',{'WP4'})
xline(time(2792),'--b',{'WP5'})
xline(time(2971),'--b',{'WP6'})
title('Yaw Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
ylim([-0.2 0.15])
xlim([0 47])
xticks(0:5:50)

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')


clear;          clc;          close all

load("8_11_Waypoint_Trial_NOVERT.mat")

waypoint1 = horzcat(waypoint_itr{1});
waypoint2 = horzcat(waypoint_itr{2});
waypoint3 = horzcat(waypoint_itr{3});
waypoint4 = horzcat(waypoint_itr{4});
waypoint5 = horzcat(waypoint_itr{5});
waypoint6 = horzcat(waypoint_itr{6});

start = 0;
first = 3.442
second = 6.292;
```

```
third = 9.4;
fourth = 12.21;
fifth = 14.47;
six = 22.14;
stop = six;

waypoints = horzcat(waypoint_itr{:});

xaxis = waypoints(1,:);
yaxis = waypoints(2,:);
zaxis = waypoints(3,:);
roll = waypoints(4,:);
pitch = waypoints(5,:);
yaw = waypoints(6,:);
timeD = waypoints(7,:);
timeDe = diff(timeD);
time =cumsum(timeDe);
```

Desires Way Points for PMD

```
figure()
x = [0 1.5 -1 -1.5 0.5 1.0 0];
y = [0 1.5 1.0 -1.5 -0.5 1.0 0];
z = [0 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8];
plot(x,y,'LineWidth',1.5)
ax =gca;
set(gca, 'FontName', 'Times New Roman')
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
hold on

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','r')
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
grid on
ylim([-2 2])
xlim([-2 2])
title('Desired Waypoints for the PMD','FontSize',18,"FontWeight","bold")
xlabel('X (m)')
ylabel('Y (m)')
```

3D View of PMD

```
figure()
labels1 = {'.','1','2','3','4','5','6'};
textscatter3(x,y,z,labels1,"FontSize",14)
hold on
plot3(0,0,0+.05,'p','MarkerFaceColor','b','MarkerSize',12)
```

```
plot3(goalDesired(1,1),goalDesired(2,1),goalDesired(3,1)+.05,'o','MarkerFaceColor','b')
plot3(goalDesired(1,2),goalDesired(2,2),goalDesired(3,2)-.05,'o','MarkerFaceColor','b')
plot3(goalDesired(1,3),goalDesired(2,3),goalDesired(3,3)-.05,'o','MarkerFaceColor','b')
plot3(goalDesired(1,4),goalDesired(2,4),goalDesired(3,4)-.05,'o','MarkerFaceColor','b')
plot3(goalDesired(1,5),goalDesired(2,5),goalDesired(3,5)-.05,'o','MarkerFaceColor','b')
plot3(goalDesired(1,6),goalDesired(2,6),goalDesired(3,6)-.05,'o','MarkerFaceColor','b')
plot3(xaxis,yaxis,zaxis,'r','LineWidth',1.5)
plot3([0 0 0.2037],[0;0;0.09335],[0 0 -0.8172],'r','LineWidth',1.5)
title('3D View of the PMD Experimental Flight Path','FontSize',18,''FontWeight'',''bold'')
grid on
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')
yticks(-1 : 1 : 2);
view([-50.981 46.285])
set(gca, 'Zdir','reverse')
set(gca, 'Xdir','reverse')
set(gca, 'FontName', 'Times New Roman')
ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
ax.ZAxis.FontSize = 14;
```

XY Response for the PMD

```
figure()

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
hold on
plot(xaxis(1,:), yaxis(1,:),'r','LineWidth',1.5)
grid on
title('XY View of the PMD Experimental Flight Path','FontSize',18,''FontWeight'',''bold'')

xlabel('X (m)')
ylabel('Y (m)')
grid on
ylim([-2 2])
xlim([-2 2])

ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
set(gca, 'FontName', 'Times New Roman')
```

X Response

```
figure()
set(gca, 'Zdir','normal')
xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
line([start, first], [xyDesired(1,1),xyDesired(1,1)],'Color','b','LineWidth',1.5);
line([first, second], [xyDesired(1,2),xyDesired(1,2)],'Color','b','LineWidth',1.5);
line([second, third], [xyDesired(1,3),xyDesired(1,3)],'Color','b','LineWidth',1.5);
line([third, fourth], [xyDesired(1,4),xyDesired(1,4)],'Color','b','LineWidth',1.5);
line([fourth, fifth], [xyDesired(1,5),xyDesired(1,5)],'Color','b','LineWidth',1.5);
line([fifth, six], [xyDesired(1,6),xyDesired(1,6)],'Color','b','LineWidth',1.5);
plot(time,xaxis(2:end),'r','LineWidth',1.5)

grid on
title('X Axis Response','FontSize',32,''FontWeight'',''bold'')
xlabel('Time (s)')
ylabel('X  (m)')
ylim([-2 2])
xlim([0 22.5])
xticks(0:5:22)
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-1.5 : 0.5 : 1.5);
```

Y Response

```
figure()

xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
line([start, first], [xyDesired(2,1),xyDesired(2,1)],'Color','b','LineWidth',1.5)
line([first, second], [xyDesired(2,2),xyDesired(2,2)],'Color','b','LineWidth',1.5)
```

```
line([second, third], [xyDesired(2,3),xyDesired(2,3)],'Color','b','LineWidth',1.5)
line([third, fourth], [xyDesired(2,4),xyDesired(2,4)],'Color','b','LineWidth',1.5)
line([fourth, fifth], [xyDesired(2,5),xyDesired(2,5)],'Color','b','LineWidth',1.5)
line([fifth, six], [xyDesired(2,6),xyDesired(2,6)],'Color','b','LineWidth',1.5)
plot(time,yaxis(2:end),'r','LineWidth',1.5)

grid on
title('Y Axis Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Y (m)')
ylim([-2 2])
xlim([0 22.5])
xticks(0:5:22)
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-2 : 0.5 : 2);
```

Z Response

```
figure()

set(gca, 'Ydir','reverse')
xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
line([start, time(665)],     [vertDesired(3,1),vertDesired(3,1)],'Color','b','LineWidth',1.5)
line([time(665), time(789)], [-0.05,-0.05],'Color','b','LineWidth',1.5)

plot(time,zaxis(2:end),'r','LineWidth',1.5)

grid on
title('Z Axis Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Z (m)')

xlim([0 22.5])
xticks(0:5:22)
ylim([-1 0])
```

```
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
```

Roll Response

```
figure()

xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
plot(time,roll(2:end),'r','LineWidth',1.5)
grid on
title('Roll Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
xlim([0 22.5])
xticks(0:5:22)
yticks(-.25:.1:.2)

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
set(gca, 'YDir','normal')
```

Pitch Response

```
figure()

xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
plot(time,pitch(2:end),'r','LineWidth',1.5)
grid on
title('Pitch Response','FontSize',32,"FontWeight","bold")
```

```
xlabel('Time (s)')
ylabel('Angle (deg)')
xlim([0 22.5])
xticks(0:5:22)
yticks(-.2:.1:.25)

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
```

Yaw Response

```
figure()

xline(time(160),'--b',{'WP1'})
hold on
xline(time(290),'--b',{'WP2'})
xline(time(432),'--b',{'WP3'})
xline(time(561),'--b',{'WP4'})
xline(time(665),'--b',{'WP5'})
xline(time(785),'--b',{'WP6'})
plot(time,yaw(2:end),'r','LineWidth',1.5)
grid on
title('Yaw Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
xlim([0 22.5])
xticks(0:5:22)
yticks(-.25:.1:.1)

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. SIMULINK PLOT SCRIPT OF TRIAL FLIGHTS

```
clear;  clc;    close all

load("PMD_Simulnk_Waypoints_good1.mat")
timeV = time*Ts;

figure()
x = [0 1.5 -1 -1.5 0.5 1.0 0];
y = [0 1.5 1.0 -1.5 -0.5 1.0 0];
z = [0 -0.8 -1.2 -0.8 -1.2 -0.8 -1.2];

plot(x,y,'LineWidth',1.5)
ax =gca;
set(gca, 'FontName', 'Times New Roman')
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
hold on

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','r')
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
grid on
ylim([-2 2])
xlim([-2 2])
title('Desired Waypoints for the PMD','FontSize',18,"FontWeight","bold")
xlabel('X (m)')
ylabel('Y (m)')

3D View of the Flight path
figure()
labels1 = {'.','1','2','3','4','5','6'};
textscatter3(x,y,z,labels1,"FontSize",14)
hold on
grid on
plot3(0,0,0,'p','MarkerFaceColor','b','MarkerSize',12)
plot3(goalDesired(1,1),goalDesired(2,1),goalDesired(3,1)+.08,'o','MarkerFaceColor','b')
plot3(goalDesired(1,2),goalDesired(2,2),goalDesired(3,2)+.08,'o','MarkerFaceColor','b')
plot3(goalDesired(1,3),goalDesired(2,3),goalDesired(3,3)+.08,'o','MarkerFaceColor','b')
plot3(goalDesired(1,4),goalDesired(2,4),goalDesired(3,4)+.08,'o','MarkerFaceColor','b')
plot3(goalDesired(1,5),goalDesired(2,5),goalDesired(3,5)+.08,'o','MarkerFaceColor','b')
plot3(goalDesired(1,6),goalDesired(2,6),goalDesired(3,6)+.08,'o','MarkerFaceColor','b')
plot3(posXY(:,1),posXY(:,2),posZ,'r','LineWidth',1.5)
title('3D View of the PMD Simulink Flight Path','FontSize',18,"FontWeight","bold")
```

```
xlabel('X (m)')
ylabel('Y (m)')
zlabel('Z (m)')
zlim([-1.5 0])
ylim([-2 2])
xlim([-2 2])
yticks(-1 : 1 : 2);
view([-59.37 20.66])

set(gca, 'Zdir','reverse')
set(gca, 'Xdir','reverse')
set(gca, 'FontName', 'Times New Roman')
ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
ax.ZAxis.FontSize = 14;
```

XY Response of the PMD

```
figure()

labels = {'Start/End','WP 1','WP 2','WP 3','WP 4','WP 5','Start/End'};
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
text(x,y,labels,'VerticalAlignment','bottom','HorizontalAlignment','right','FontSize',14)
hold on
plot(posXY(:,1), posXY(:,2),'r','LineWidth',1.5)
title('XY View of the PMD Simulnk Flight Path','FontSize',18,"FontWeight","bold")
xlabel('X (m)')
ylabel('Y (m)')
grid on
ylim([-2 2])
xlim([-2 2])

ax =gca;
ax.XAxis.FontSize = 14;
ax.YAxis.FontSize = 14;
set(gca, 'FontName', 'Times New Roman')
```

X Response

```
figure()

line([0, timeV(1000)], [0.01,0.01],'Color','b','LineWidth',1.5)
hold on
line([timeV(1001), timeV(2281)],
[goalDesired(1,1),goalDesired(1,1)],'Color','b','LineWidth',1.5)
```

```
line([timeV(2281), timeV(3844)],
[goalDesired(1,2),goalDesired(1,2)],'Color','b','LineWidth',1.5)
line([timeV(3844), timeV(4965)],
[goalDesired(1,3),goalDesired(1,3)],'Color','b','LineWidth',1.5)
line([timeV(4965), timeV(6096)],
[goalDesired(1,4),goalDesired(1,4)],'Color','b','LineWidth',1.5)
line([timeV(6096), timeV(6800)],
[goalDesired(1,5),goalDesired(1,5)],'Color','b','LineWidth',1.5)
line([timeV(6800), timeV(7949)], [0.01,0.01],'Color','b','LineWidth',1.5)
xline(timeV(1030),'--b',{'Start'})
xline(timeV(2281),'--b',{'WP1'})
xline(timeV(3844),'--b',{'WP2'})
xline(timeV(4965),'--b',{'WP3'})
xline(timeV(6096),'--b',{'WP4'})
xline(timeV(6800),'--b',{'WP5'})
xline(timeV(7949),'--b',{'WP6'})
plot(timeV,posXY(:,1),'r','LineWidth',1.5)

grid on
title('X Axis Response','FontSize',32,''FontWeight'',''bold'')
xlabel('Time (s)')
ylabel('X  (m)')
ylim([-2 2])
xlim([0 41])
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-2 : 1 : 2);
```

Y Response

```
figure()

line([0, timeV(1000)], [0.01,0.01],'Color','b','LineWidth',1.5)
hold on
line([timeV(1001), timeV(2281)],
[goalDesired(2,1),goalDesired(2,1)],'Color','b','LineWidth',1.5)
line([timeV(2281), timeV(3844)],
[goalDesired(2,2),goalDesired(2,2)],'Color','b','LineWidth',1.5)
line([timeV(3844), timeV(4965)],
[goalDesired(2,3),goalDesired(2,3)],'Color','b','LineWidth',1.5)
line([timeV(4965), timeV(6096)],
[goalDesired(2,4),goalDesired(2,4)],'Color','b','LineWidth',1.5)
```

```
line([timeV(6096), timeV(6800)],
[goalDesired(2,5),goalDesired(2,5)],'Color','b','LineWidth',1.5)
line([timeV(6800), timeV(7949)], [0.01,0.01],'Color','b','LineWidth',1.5)
xline(timeV(1030),'--b',{'Start'})
xline(timeV(2281),'--b',{'WP1'})
xline(timeV(3844),'--b',{'WP2'})
xline(timeV(4965),'--b',{'WP3'})
xline(timeV(6096),'--b',{'WP4'})
xline(timeV(6800),'--b',{'WP5'})
xline(timeV(7949),'--b',{'WP6'})
plot(timeV,posXY(:,2),'r','LineWidth',1.5)

grid on
title('Y Axis Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Y (m)')
ylim([-2 2])
xlim([0 41])
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
yticks(-2 : 1 : 2);
```

Z Response

```
figure()

line([0, timeV(1030)], [0.01,0.01],'Color','b','LineWidth',1.5)
hold on
line([timeV(1030), timeV(2281)],
[goalDesired(3,1),goalDesired(3,1)],'Color','b','LineWidth',1.5)
line([timeV(2281), timeV(3844)],
[goalDesired(3,2),goalDesired(3,2)],'Color','b','LineWidth',1.5)
line([timeV(3844), timeV(4965)],
[goalDesired(3,3),goalDesired(3,3)],'Color','b','LineWidth',1.5)
line([timeV(4965), timeV(6096)],
[goalDesired(3,4),goalDesired(3,4)],'Color','b','LineWidth',1.5)
line([timeV(6096), timeV(6800)],
[goalDesired(3,5),goalDesired(3,5)],'Color','b','LineWidth',1.5)
line([timeV(6800), timeV(7949)],
[goalDesired(3,6),goalDesired(3,6)],'Color','b','LineWidth',1.5)
line([timeV(7949), timeV(8164)], [0.01,0.01],'Color','b','LineWidth',1.5)
xline(timeV(1030),'--b',{'Start'})
```

```
xline(timeV(2281),'--b',{'WP1'})
xline(timeV(3844),'--b',{'WP2'})
xline(timeV(4965),'--b',{'WP3'})
xline(timeV(6096),'--b',{'WP4'})
xline(timeV(6800),'--b',{'WP5'})
xline(timeV(7949),'--b',{'WP6'})
plot(timeV, posZ,'r','LineWidth',1.5)

grid on
title('Z Axis Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Z (m)')
ylim([-1.5 0.1])
hold off

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
set(gca, 'Ydir','reverse')
xlim([0 41])
```

Roll Response

```
figure()

xline(timeV(1030),'--b',{'WP1'})
hold on
xline(timeV(2281),'--b',{'WP2'})
xline(timeV(3844),'--b',{'WP3'})
xline(timeV(4965),'--b',{'WP4'})
xline(timeV(6096),'--b',{'WP5'})
xline(timeV(6800),'--b',{'WP6'})
xline(timeV(7949),'--b',{'Land'})
plot(timeV,roll,'r','LineWidth',1.5)
grid on
title('Roll Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
set(gca, 'YDir','normal')
xlim([0 41])
```

Pitch Response

```
figure()
plot(timeV,pitch,'r','LineWidth',1.5)
grid on
xline(timeV(1030),'--b',{'WP1'})
hold on
xline(timeV(2281),'--b',{'WP2'})
xline(timeV(3844),'--b',{'WP3'})
xline(timeV(4965),'--b',{'WP4'})
xline(timeV(6096),'--b',{'WP5'})
xline(timeV(6800),'--b',{'WP6'})
xline(timeV(7949),'--b',{'Land'})
title('Pitch Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')

ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
xlim([0 41])
```

Yaw Response

```
figure()

plot(timeV,yaw,'r','LineWidth',1.5)
xline(timeV(1030),'--b',{'WP1'})
hold on
xline(timeV(2281),'--b',{'WP2'})
xline(timeV(3844),'--b',{'WP3'})
xline(timeV(4965),'--b',{'WP4'})
xline(timeV(6096),'--b',{'WP5'})
xline(timeV(6800),'--b',{'WP6'})
xline(timeV(7949),'--b',{'Land'})
grid on
title('Yaw Response','FontSize',32,"FontWeight","bold")
xlabel('Time (s)')
ylabel('Angle (deg)')
ax =gca;
ax.XAxis.FontSize = 18;
ax.YAxis.FontSize = 18;
set(gca, 'FontName', 'Times New Roman')
xlim([0 41])
```

APPENDIX I. ERROR PLOTS AND CALCULATIONS

```
clear;  clc;    close all

load("PMD_Simulnk_Waypoints_good1.mat")
TEr = [XYError PitchRollZError PitchRollError];
timeV = time*Ts;


xWPErrors =
[TEr(962,1);TEr(2190,1);TEr(3755,1);TEr(4970,1);TEr(6158,1);TEr(6800,1);TEr(7949,1
)];
yWPErrors =
[TEr(962,2);TEr(2286,2);TEr(3844,2);TEr(4991,2);TEr(6045,2);TEr(6801,2);TEr(7949,2
)];
zWPErrors =
[TEr(1135,3);TEr(2281,3);TEr(3844,3);TEr(4965,3);TEr(6096,3);TEr(6800,3);TEr(7949,
3)];
rWPErrors =
[TEr(962,4);TEr(2281,4);TEr(3909,4);TEr(4965,4);TEr(6096,4);TEr(6800,4);TEr(7949,4
)];
pWPErrors =
[TEr(962,5);TEr(2281,5);TEr(3909,5);TEr(4965,5);TEr(6096,5);TEr(6800,5);TEr(7949,5
)];
yaWPErrors = [norm(rWPErrors(1)+pWPErrors(1)) norm(rWPErrors(2)+pWPErrors(2))
norm(rWPErrors(3)+pWPErrors(3)) norm(rWPErrors(4)+pWPErrors(4))
norm(rWPErrors(5)+pWPErrors(5)) norm(rWPErrors(6)+pWPErrors(6))
norm(rWPErrors(7)+pWPErrors(7))]';

PosErrors = abs([xWPErrors yWPErrors zWPErrors yaWPErrors]);
yawD = [10^-5 10^-5 10^-5 10^-5 10^-5 10^-5 10^-5];

goalDesired2 = [1.5    1.5      -0.8     1
            -1     1        -1.2     1
            -1.5  -1.5     -0.8     1
            0.5   -0.5     -1.2     1
            1      1        -0.8     1
            0.0001       0.0001 -1.2     1
            0.0001       0.0001 0.0001 1];

start1 = (PosErrors(1,:)./abs(goalDesired2(1,:)))*100;
WP1 = (PosErrors(2,:)./abs(goalDesired2(2,:)))*100;
WP2 = (PosErrors(3,:)./abs(goalDesired2(3,:)))*100;
WP3 = (PosErrors(4,:)./abs(goalDesired2(4,:)))*100;
```

```
WP4 = (PosErrors(5,:)./abs(goalDesired2(5,:)))*100;
WP5 = (PosErrors(6,:)./abs(goalDesired2(6,:)))*100;
WP6 = (PosErrors(7,:)./abs(goalDesired2(7,:)))*100;

totalEr = abs([sum(start1)/4 sum(WP1)/4 sum(WP2)/4 sum(WP3)/4 sum(WP4)/4
sum(WP5)/4 sum(WP6)/4]);
start = [start1 totalEr(1)]';
WPs1 = [WP1 totalEr(2)]';
WPs2 = [WP2 totalEr(3)]';
WPs3 = [WP3 totalEr(4)]';
WPs4 = [WP4 totalEr(5)]';
WPs5 = [WP5 totalEr(6)]';
WPs6 = [WP6 totalEr(7)]';



ErrorState = {'X','Y','Z','Yaw','TotalError'}';
table(ErrorState,start,WPs1,WPs2,WPs3,WPs4,WPs5,WPs6)

clear;  clc;    close all

load("8_11_Waypoint_Trial_NOVERT.mat")

totalError = horzcat(totalError_itr{:});

goalDesired2 = [1.5       1.5      -0.8  1
            -1        1        -1.2  1
            -1.5  -1.5     -0.8     1
            0.5       -0.5  -1.2     1
            1         1        -0.8  1
            0.0001        0.0001 -1.2      1]';


WP1 = abs(horzcat(totalError_itr{1}));
WP2 = abs(horzcat(totalError_itr{2}));
WP3 = abs(horzcat(totalError_itr{3}));
WP4 = abs(horzcat(totalError_itr{4}));
WP5 = abs(horzcat(totalError_itr{5}));
WP6 = abs(horzcat(totalError_itr{6}));

WPs1 = (WP1./abs(goalDesired2(:,1)))*100;
WPs2 = (WP2./abs(goalDesired2(:,2)))*100;
WPs3 = (WP3./abs(goalDesired2(:,3)))*100;
WPs4 = ([4.47123243; 9.41566228866577; 2.70632721019835; 8.77375721931457]);
WPs5 = (WP5./abs(goalDesired2(:,5)))*100;
```

```
WPs6 = (WP6./abs(goalDesired2(:,6)))*100;

norm(sum(WPs1))

sum(WPs1)

TotalErrors = [norm(sum(WPs1))/4 norm(sum(WPs2))/4 norm(sum(WPs3))/4
norm(sum(WPs4))/4 norm(sum(WPs5))/4 norm(sum(WPs6))/4];

WPs1 = abs([WPs1 ;TotalErrors(1)]);
WPs2 = abs([WPs2 ;TotalErrors(2)]);
WPs3 = abs([WPs3 ;TotalErrors(3)]);
WPs4 = abs([WPs4 ;TotalErrors(4)]);
WPs5 = abs([WPs5 ;TotalErrors(5)]);
WPs6 = abs([WPs6 ;TotalErrors(6)]);

ErrorState = {'X','Y','Z','Yaw','TotalError'}';
table(ErrorState,WPs1,WPs2,WPs3,WPs4,WPs5,WPs6)

clear;  clc;    close all

load("8_11_Waypoint_Trial1_VERTFULL.mat")

totalError = horzcat(totalError_itr{:});

goalDesired2 = [1.5    1.5      -0.8     1
            -1     1        -1.2     1
            -1.5  -1.5     -0.8     1
            0.5   -0.5     -1.2     1
            1      1        -0.8     1
            0.0001       0.0001 -1.2     1]';


WP1 = abs(horzcat(totalError_itr{1}));
WP2 = abs(horzcat(totalError_itr{2}));
WP3 = abs(horzcat(totalError_itr{3}));
WP4 = abs(horzcat(totalError_itr{4}));
WP5 = abs(horzcat(totalError_itr{5}));
WP6 = abs(horzcat(totalError_itr{6}));

WPs1 = (WP1./abs(goalDesired2(:,1)))*100;
WPs2 = (WP2./abs(goalDesired2(:,2)))*100;
WPs3 = (WP3./abs(goalDesired2(:,3)))*100;
WPs4 = (WP4./abs(goalDesired2(:,4)))*100;
WPs5 = (WP5./abs(goalDesired2(:,5)))*100;
```

```
WPs6 = (WP6./abs(goalDesired2(:,6)))*100;

TotalErrors = [norm(sum(WPs1))/4 norm(sum(WPs2))/4 norm(sum(WPs3))/4
norm(sum(WPs4))/4 norm(sum(WPs5))/4 norm(sum(WPs6))/4];

WPs1 = abs([WPs1 ;TotalErrors(1)]);
WPs2 = abs([WPs2 ;TotalErrors(2)]);
WPs3 = abs([WPs3 ;TotalErrors(3)]);
WPs4 = abs([WPs4 ;TotalErrors(4)]);
WPs5 = abs([WPs5 ;TotalErrors(5)]);
WPs6 = abs([WPs6 ;TotalErrors(6)]);

ErrorState = {'X','Y','Z','Yaw','TotalError'}';
table(ErrorState,WPs1,WPs2,WPs3,WPs4,WPs5,WPs6)
```

LIST OF REFERENCES

[1] R. Debevec, “A smart UAV platform for railroad inspection,” M.S. thesis, Dept. of Mechanical and Aerospace Engineering, CSUN, Northridge, CA, 2019. [Online]. Available: http://purl.fcia.edu/fcla/etd/CFE0007623

[2] R. Ji and J. Ma, “Mathematical modeling and analysis of a quadrotor with tilting propellers,” *2018 37th Chinese Control Conference (CCC)*, pp. 1718–1722, July 2018.

[3] S. Bouabdallah, A. Noth, and R. Siegwart, “PID vs LQ control techniques applied to an indoor micro quadrotor,” *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, pp. 2451–2456, vol.3, October 2004.

[4] M. Peters, “The engineering analysis and design of the aircraft dynamics model for the FAA Target Generation Facility,” in *Contract Project, Simulation Branch, Laboratory Services Division*, William J. Hughes Technical Center, Atlantic City, NJ,2012. [Online]. Available: https://www.faa.gov/about/officeorg /headquarters_offices/ang/offices/tc/about/campus/faa_host/labs/tgf/media/aircraft dynamicsmodel.pdf

[5] R. L. Allen, “Quadrotor intercept trajectory planning and simulation,” M.S. thesis, Dept. of Electrical Engineering, NPS, Monterey, CA, 2017. [Online]. Available: https://hdl. handle.net/10945/55627

[6] A. R. Babaei, M. Mortazavi, and M. H. Moradi, “Classical and fuzzy-genetic autopilot design for unmanned aerial vehicles,” *Applied Soft Computing Journal*, vol. 11, no. 1, pp. 365–372, 2011.

[7] G. E. Setyawan, W. Kurniawan, and A. C. L. Gaol, “Linear quadratic regulator controller (LQR) for AR. drone’s safe landing,” *2019 International Conference on Sustainable Information Engineering and Technology (SIET)*, Lombok, Indonesia, 2019, pp. 228–233, Sept. 2019.

[8] N. S. Nise, *Control Systems Engineering*, 6th Ed., Hoboken, NJ: JohnWiley & Sons, 2011.

[9] R. Kumar, “Comparison among some well-known control schemes with different tuning methods,” *Journal of Applied Research and Technology*, vol. 13, no. 3, pp. 409–415, June 2015. [Online]. Available: https://www.sciencedirect.com/science /article/pii/ S1665642315000358

[10] “Rigid body dynamics,” class notes for ME 3801, Dynamics and Control of Marine and Autonomous Vehicles, Dept. of Mechanical Engineering, Naval Postgraduate School, Monterey, CA, USA, summer 2019.

[11] Parrot, “Parrot Mambo Fly.” 2020. [Online]. Available: https://www.parrot.com/ us/ drones

[12] “Equations of motion for three dimensional rigid bodies,” class notes for EC4330, Navigation, Missile, and Avionics Systems, Dept. of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA, USA, spring 2020.

[13] Wikipedia, “Coordinate system” 2020. [Online]. Available: https://en.wikipedia. org/wiki/ Coordinate_system.

[14] MathWorks, “quat2eul,” 2020. [Online]. Available: https://www. mathworks.com/help/robotics/ref/quat2eul.html

[15] MathWorks, “What is MATLAB?” 2020. [Online]. Available: https://www. mathworks.com/discovery/what-is-MATLAB.html

[16] MathWorks, “What is Simulink?” 2020. [Online]. Available: https://www. mathworks. com/videos/simulink-overview-61216.html

[17] OptiTrack, “Motive: Tracker motion capture & 6 DOF object tracking.” 2020. [Online]. Available: https://www.optitrack.com/products/motive/tracker.html

[18] OptiTrack, “Data types.” 2020. [Online]. Available: https://v22.wiki.optitrack. com/index.php?title=File:AssetsPane_RecordSolved.png

[19] MathWorks, “System modeling and simulation.” 2020. [Online]. Available: https://www.mathworks.com/solutions/system-design-simulation.html

[20] MathWorks, “Hardware support.” 2020. [Online]. Available: https://www. mathworks.com/hardware-support/parrot-drone-MATLAB.html

[21] MathWorks, “Drone navigation.” 2020. [Online]. Available: https://www. mathworks.com/help/supportpkg/parrotio/drone-navigation.html

[22] OptiTrack, “About OptiTrack.” 2020. [Online]. Available: https://www. optitrack.com/ about/

[23] OptiTrack, “Support: General FAQs.” 2020. [Online]. Available: https:// optitrack.com/ support/faq/general.html

[24] OptiTrack, “Prime X 13W.” 2020. [Online]. Available: https://optitrack.com/ products/ primex-41/

[25] MathWorks, “Simulink support package for Parrot minidrones.” 2020. [Online]. Available: https://www.mathworks.com/MATLABcentral/fileexchange/ 63318-simulink-support-package-for-parrot-minidrones

[26] K. Kyunam, S. Rahili, X. Shi, and S. Chung, “Controllability and design of unmanned multirotor aircraft robust to rotor failure,” in *Session: Design of Unmanned Aircraft Systems II, California Institute of Technology 2019, January 2019, Pasadena, CA*, [Online]. Available: https:// doi.org/10.2514/6.2019-1787

[27] S. Zouaoui, E. Mohamed, and K. Bendine, “Easy tracking of UAV using PID controller,” *Period. Polytech. Transp. Eng.,* vol. 47, no. 3, pp. 171–177, Jan. 2019.

[28] A. Shekhar and A. Sharma, “Review of model reference adaptive control,” *International Conference on Information, Communication, Engineering and Technology (ICICET)*, Pune, 2018, pp. 1–5, Aug. 2018.

[29] V.I. Utkin, “Sliding Mode Control: Mathematical tools, design and applications.” in *Nistri P., Stefani G. (eds) Nonlinear and Optimal Control Theory, Lecture Notes in Mathematics 2008, Springer, Berlin, Heidelberg*, vol. 1932, pp. 324–328 [Online]. Available: https://doi.org/10.1007/978-3-540-77653-6_5

[30] MathWorks, “Follow set of waypoints or follow orbit using Parrot minidrone.” 2020. [Online]. Available: https://www.mathworks.com/help/ supportpkg/parrot/ref/follow-waypoints-parrot-drone.html

[31] C. Wackerman, “Operational assimilation into UUV and UAV observations,” M.S. thesis, Dept. of Operations and Naval Research, NPS, Monterey, CA, 2019. [Online]. Available: https://hdl.handle.net/10945/63474

[32] “Optimal controls class,” class notes for EC 3320, Optimal Controls Systems, Dept. of Electrical Engineering, Naval Postgraduate School, Monterey, CA, USA, summer 2019.

[33] MathWorks, “Waypoint follower.” 2020. [Online]. Available: https://www. mathworks.com/help/robotics/ref/ waypointfollower.html

[34] Office of Naval Research, “Operational assimilation into UUV and UAV operations.” 2018. [Online]. Available: https://www.onr.navy.mil/en/-/ media/Files/Annual-Reports/Coastal-Geosciences/FY08/cgwacker.

[35] Office of Naval Research, “Autonomous, swarming UAV fly into the future.” 2015. [Online]. Available: https://www.onr.navy.mil/en/Media-Center/Press-Releases/2015/LOCUST-low-cost-UAV-swarm-ONR

[36] K. Bernauw, “Drones: The emerging era of unmanned civil aviation.” *Zbornik Pravnog Fakultetau Zagrebu*, vol. 66(2/3), pp. 223, Feb. 2016.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California